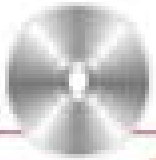
Naucz się tworzyć efektowne aplikacje internetowe dzięki możliwościom Adobe Flex 4!

- Tworzenie własnych komponentów, nadawanie im stylów i skórek
- Praca nad aplikacjami z wykorzystaniem środowiska Flash Builder
- Wydajny, zorientowany obiektowo język ActionScript 3.0



ADOBE® FLEX® 4

OFICJALNY PODRĘCZNIK

Michael Labriola, Jeff Tapper, Matthew Boles
Wstęp: Matt Chotin, menedżer produktu

Adobe Flex 4. Oficjalny podręcznik

Autorzy: Michael Labriola, Jeff Tapper, Matthew Boles
Tłumaczenie: Aleksander Lamża
ISBN: 978-83-246-2884-1
Tytuł oryginału: Adobe Flex 4: Training from the Source, Volume 1
Format: B5, stron: 464

Naucz się tworzyć efektowne aplikacje internetowe dzięki możliwościom Adobe Flex 4!

- Tworzenie własnych komponentów, nadawanie im stylów i skórek
- Praca nad aplikacjami z wykorzystaniem środowiska Flash Builder
- Wydajny, zorientowany obiektowo język ActionScript 3.0

Adobe Flex 4 to najnowszy zestaw technologii służących do tworzenia bogatych aplikacji internetowych. Ta wydajna i bezpłatna struktura programistyczna to istne novum na rynku programów webowych. Flex pozwala na tworzenie prawdziwie dynamicznych i interaktywnych aplikacji o olbrzymich możliwościach. A to wszystko bez konieczności rezygnowania ze wspaniałej grafiki!

Dzięki książce „Adobe Flex 4. Oficjalny podręcznik" stopniowo, krok po kroku, nauczysz się tworzyć aplikacje w językach MXML oraz ActionScript 3.0. Znajdziesz tu omówienie nowości wprowadzonych we Fleksie 4, takich jak komponenty Spark, nowe, zaawansowane techniki tworzenia skórek oraz wiele innych. Poznasz możliwości pracy w środowisku Flex Builder oraz dowiesz się, jak tworzyć interfejsy użytkownika. Zapoznasz się z wiedzą na temat kontrolek, obsługi zdarzeń, stosowania techniki „przeciągnij i upuść", tworzenia nawigacji czy też zmiany wyglądu Twojej aplikacji za pomocą stylów. Pamiętaj, że masz przed sobą najlepszy podręcznik technologii Flex, napisany przez zespół specjalistów posiadających olbrzymie doświadczenie w prowadzeniu kursów programowania, konsultacji i wszelkich działań wspomagających programistów i projektantów.

- Flash Builder
- Tworzenie interfejsu użytkownika
- Stosowanie kontrolek
- Obsługa zdarzeń
- ActionScript i MXML
- Tworzenie komponentów
- Siatki danych
- Rendering elementów
- Stosowanie mechanizmu „przeciągnij i upuść"
- Style i skórki

Skorzystaj z niesamowitych możliwości Fleksa 4 i stwórz swoją wyjątkową aplikację internetową!

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

Spis treści

Tematyka lekcji 3.

Podczas tej lekcji będziesz:

- ✧ używał kontenerów;
- ✧ tworzył aplikację w trybie edycji kodu;
- ✧ pracował z layoutami opartymi na więzach;
- ✧ pracował ze stanami widoku;
- ✧ sterował stanami widoku;
- ✧ tworzył aplikację w trybie projektowania;
- ✧ w razie potrzeby poddawał kod refaktoryzacji.

Przewidywany czas lekcji

Ukończenie tej lekcji zajmuje około 1 godziny i 30 minut.

3 Projektowanie interfejsu użytkownika

Każda aplikacja musi posiadać interfejs użytkownika. Jedną z największych zalet programu Adobe Flash Builder 4 jest to, że bardzo ułatwia on projektowanie interfejsu aplikacji. Podczas tej lekcji zostanie przedstawionych wiele kontenerów używanych we Fleksie, opiszemy różnice między nimi oraz metody wykorzystywania ich podczas tworzenia interfejsów użytkownika własnych projektów. Zastosowanie stanów pozwala projektantowi na tworzenie aplikacji dynamicznie dostosowujących się do działań użytkowników.

Interfejs użytkownika sklepu internetowego

Layout aplikacji

Większość działań związanych z ustalaniem położenia komponentów we Fleksie jest przeprowadzana z wykorzystaniem kontenerów i obiektów layoutu.

Posłużmy się na chwilę analogią do sprzętów kuchennych. Wyobraź sobie, że kontener jest robotem kuchennym bez ostrzy. Dostępnych jest wiele robotów kuchennych oferujących różne funkcje, a Twoim zadaniem jest wybranie tego, który najlepiej sprawdzi się w określonych zastosowaniach.

O obiektach layoutu możesz myśleć jak o ostrzach, które mogą służyć do siekania, cięcia na plasterki, w kostkę itp. Żaden z tych elementów (robot kuchenny i ostrza) nie jest użyteczny w pojedynkę — dopiero kiedy się je połączy, stają się przydatnym narzędziem. Ta sama zasada dotyczy kontenerów i obiektów układu graficznego.

Kontenery

Z technicznego punktu widzenia kontenery są specjalnym typem komponentów, które zawierają i grupują inne elementy. Elementy te zwykle określa się ogólnie mianem *potomków* lub, bardziej szczegółowo, jako elementy layoutu (co odpowiada takim komponentom, jak przyciski, pola wyboru itp.) oraz elementy graficzne (takie jak prostokąty, koła itp.). Mimo że kontenery potrafią grupować i utrzymywać wszystkie elementy razem, nie znają ich położenia i kolejności, w jakiej mają pojawić się na ekranie. Przy wyborze odpowiedniego kontenera trzeba się kierować wieloma kryteriami, jednak najistotniejsze są możliwości zastosowania skórek.

Dzięki stosowaniu *skórek* możliwa jest zmiana wyglądu komponentów. W przypadku kontenerów można mówić o wyglądzie tła, krawędzi, zastosowaniu cieni itd. Do niektórych kontenerów można zastosować skórki, czyli można zdefiniować ich wygląd, z kolei inne służą tylko i wyłącznie do grupowania potomków i jako takie nie są widoczne.

Typy kontenerów

Kontener	Opis
`Group`	Najprostszy typ kontenera we Fleksie 4. Może przechowywać potomków, ale nie ma graficznej reprezentacji
`SkinnableContainer`	Oferuje te same funkcjonalności co kontener `Group`, ale dodatkowo istnieje możliwość zdefiniowania jego wyglądu
`BorderContainer`	Typ kontenera `SkinnableContainer` służący do prostego otoczenia potomków ramką
`Panel`	Typ kontenera `SkinnableContainer` otoczony ramką, który można dodatkowo wyposażyć w nagłówek i obszar przeznaczony na pasek narzędzi
`Application`	Typ kontenera `SkinnableContainer` stosowany jako główny kontener fleksowej aplikacji. Podobnie jak `Panel` może posiadać pasek narzędzi
`NavigationContent`	Specjalny typ kontenera `SkinnableContener` stosowany z kontrolkami, takimi jak `ViewStack`, które zostaną omówione w dalszej części książki

To oczywiście nie wszystkie dostępne kontenery. W kolejnych lekcjach przeczytasz jeszcze np. o `DataGroup` i `SkinnableDataContainer`, a także specjalizowanych kontenerach, takich jak np. `Form`. Wymienione tu kontenery działają jednak trochę inaczej, więc ich opis zostawimy na później, kiedy pojawi się potrzeba omówienia ich specyfiki.

Obiekty layoutu

Obiekty layoutu współpracują z kontenerami (a także innymi typami obiektów, co zostanie omówione w kolejnych lekcjach) w celu określenia sposobu grupowania elementów wyświetlanych na ekranie.

Flex 4 dostarcza wiele gotowych obiektów layoutu oraz pozwala na tworzenie własnych, umożliwiając w ten sposób pełne dopasowanie do własnych potrzeb.

Typy obiektów layoutu

Obiekt layoutu	Opis
`BasicLayout`	Pozwala na bezwzględne pozycjonowanie, co oznacza, że w przypadku każdego umieszczanego elementu trzeba określić współrzędne *x* i *y* jego położenia
`HorizontalLayout`	Ustawia wszystkich potomków w rzędzie; każdy jest wyrównany do prawej krawędzi poprzedniego
`VerticalLayout`	Ustawia wszystkich potomków w kolumnie; każdy jest wyrównany do dolnej krawędzi poprzedniego
`TileLayout`	Ustawia potomków w rzędach i kolumnach. Można określić liczbę elementów ułożonych w pionie lub poziomie, której przekroczenie powoduje dodanie kolejnego wiersza lub kolumny

Łączenie kontenerów i obiektów layoutu

Po wybraniu odpowiedniego kontenera i obiektu layoutu składa się je w MXML-u w celu osiągnięcia zamierzonego efektu. Przyjrzyj się poniższym przykładom ustalającym położenie przycisków za pomocą obiektu layoutu i właściwości `layout`.

```
<s:Group>
   <s:layout>
      <s:HorizontalLayout/>
   </s:layout>

   <s:Button label="1"/>
   <s:Button label="2"/>
   <s:Button label="3"/>
</s:Group>
```

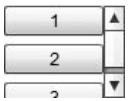

```
<s:Group>
   <s:layout>
      <s:VerticalLayout/>
   </s:layout>

   <s:Button label="1"/>
   <s:Button label="2"/>
   <s:Button label="3"/>
</s:Group>
```

Jeżeli nie określisz obiektu layoutu, zostanie zastosowany `BasicLayout`, co oznacza, że konieczne jest ustalenie współrzędnych *x* i *y* każdego przycisku. W przeciwnym razie wszystkie zostaną wyświetlone w domyślnej lokalizacji, czyli we współrzędnych (0, 0).

Przewijanie zawartości

Czasami pojawia się konieczność zapewnienia w aplikacji możliwości przewijania zawartości grupy. W poprzednich wersjach Fleksa wszystkie kontenery domyślnie posiadały tę możliwość. Mimo że takie rozwiązanie było wygodne dla programistów, wiązało się z obciążeniem każdego kontenera kodem wymaganym do obsługi przewijania, nawet jeżeli ta funkcja nie była wykorzystywana. We Fleksie 4 trzeba jawnie określić, że dany kontener można przewijać. Realizuje się to za pomocą znacznika `Scroller`, którym obejmuje się znacznik `Group`.

```
<s:Scroller height="65">
   <s:Group>
      <s:layout>
         <s:VerticalLayout/>
      </s:layout>

      <s:Button label="1"/>
      <s:Button label="2"/>
      <s:Button label="3"/>
   </s:Group>
</s:Scroller>
```



Zastosowanie tego rozwiązania nie wiąże się z niepotrzebnym wyświetlaniem pasków przewijania. Znacznik `Scroller` dodaje paski przewijania (pionowy, poziomy lub oba) w sytuacjach, gdy są niezbędne, czyli wtedy, gdy nie ma możliwości wyświetlenia elementu `Group` w pełnych wymiarach. W poprzednim przykładzie wysokość znacznika `Scroller` została ustawiona na 65 pikseli, tak aby pojawił się pionowy pasek przewijania. Jeżeli nie jest ustawiona ani wysokość, ani szerokość, Flex stara się zmieścić cały element `Group` na ekranie i dopiero wtedy, gdy się to nie udaje, wyświetlane są odpowiednie paski przewijania.

Budowa znaczników MXML

Zanim rozpoczniesz kolejne ćwiczenie, musisz poznać pewną ważną zasadę. W języku MXML *instancje klasy* są czymś innym niż *właściwości*. We fragmencie kodu z poprzedniego podrozdziału możesz znaleźć definicję przycisku. Właściwość `label` przycisku jest zdefiniowana jako atrybut znacznika `Button`:

```
<s:Button label="3"/>
```

W MXML-u można jednak zdefiniować tę samą właściwość za pomocą znaczników potomnych. W omawianym przypadku kod mógłby wyglądać następująco:

```
<s:Button>
   <s:label>3</s:label>
</s:Button>
```

Bez względu na sposób definiowania klas efekt wyświetlony na ekranie będzie identyczny. Po jakimś czasie pracy z Fleksem nauczysz się poprawnie dobierać odpowiednią składnię do sytuacji, jednak na początku może to powodować pewien zamęt.

Jak myślisz, która z przedstawionych możliwości to definicja właściwości, a która klasy? Kluczem do rozwiązania tej zagadki jest wielkość pierwszej litery nazwy znajdującej się po przestrzeni nazw (w tym przypadku po `s:`). Kiedy jest to wielka litera (tak jak *B* w nazwie *Button*), kod jest odpowiedzialny za utworzenie nowej instancji klasy. Z kolei mała litera (*l* w nazwie *label*) oznacza, że mamy do czynienia z właściwością klasy.

Przypatrz się większemu fragmentowi kodu:

```
<s:Group>
   <s:layout>
      <s:VerticalLayout/>
   </s:layout>

   <s:Button label="1"/>
   <s:Button label="2"/>
   <s:Button>
      <s:label>3</s:label>
   </s:Button>
</s:Group>
```

Litera *G* w znaczniku `<s:Group>` jest wielka, więc tworzona jest tu instancja klasy `Group`. Z kolei *l* w znaczniku `<s:layout>` jest małe, co oznacza, że jest to właściwość znacznika `Group`. Nazwa znacznika `<s:VerticalLayout>` rozpoczyna się wielką literą *V*, więc tworzona jest nowa instancja klasy `VerticalLayout`.

Gdyby przetłumaczyć kod na zdania, mógłby brzmieć tak: utwórz instancję klasy `Group`. Do właściwości `layout` tej grupy przypisz nową instancję klasy `VerticalLayout`. Do grupy dodaj trzy przyciski (klasy `Button`) z etykietami `1`, `2` i `3`.

Bardzo ważne jest, byś w pełni zrozumiał zasady opisane w tym podrozdziale. Jeżeli wszystko jest jasne, ciąg dalszy tej lekcji nie będzie dużym wyzwaniem. Jeśli jednak nie jesteś wszystkiego pewien, podczas lektury dalszej części lekcji możesz poczuć zniechęcenie.

Tworzenie layoutu sklepu internetowego

Sklep internetowy FlexGrocer jest aplikacją typu e-commerce, która umożliwia klientom dokonywanie zakupów artykułów spożywczych. W górnej części interfejsu użytkownika jest umieszczone logo sklepu oraz odnośniki dostępne w całej aplikacji. Poniżej znajduje się grupa ikon, dzięki którym użytkownik może przeglądać dostępne kategorie produktów (nabiał, mięso, owoce itd.). Pod ikonami znajduje się obszar wyświetlania informacji o produktach.

Podczas wykonywania ćwiczeń i opracowywania layoutu aplikacji skorzystasz zarówno z trybu projektowania (*Design*), jak i edycji źródła (*Source*). Tryb projektowania ma bardzo duże możliwości, ale na początku pracy z nim mogą się pojawiać problemy. Często zdarzają się trudności z prawidłowym wyrównaniem obiektów czy umieszczaniem obiektów w wybranych kontenerach. Z tego względu w ćwiczeniach znajdziesz fragmenty kodu odpowiadające określonym elementom aplikacji. Dzięki temu w razie problemów z uzyskaniem identycznego efektu co w książce możesz przełączyć się do trybu edycji źródła i wprowadzić w kodzie niezbędne poprawki.

Tworzenie zarysu layoutu w trybie edycji źródła

Pierwszy etap tworzenia layoutu wykonasz w trybie edycji źródła. Zdefiniujesz obszary aplikacji przeznaczone na logo i niektóre elementy nawigacyjne.

1. Wczytaj utworzony w poprzedniej lekcji plik *FlexGrocer.mxml*.

   Jeżeli nie ukończyłeś poprzedniej lekcji lub przygotowany kod nie działa prawidłowo, możesz zaimportować projekt *FlexGrocer.fxp* z katalogu *Lekcja03/start*. Szczegółowe informacje na temat importowania projektu znajdziesz w dodatku A. Skorzystaj z uwag tam zawartych za każdym razem, gdy opuścisz którąś lekcję lub natrafisz na problem, którego nie umiesz rozwiązać.

2. Włącz tryb edycji źródła (przycisk *Source*).

   Do przełączania między trybem projektowania (*Design*) i trybem edycji źródła (*Source*) używaj przycisków umieszczonych na pasku tytułowym w górnej części okna.

3. Usuń znacznik `Label` wyświetlający tekst „My First Flex Application”, który dodałeś w poprzedniej lekcji.

4. W miejsce usuniętego elementu wstaw otwierający i zamykający znacznik `controlBarLayout`.

   ```
   <s:controlBarLayout>
   </s:controlBarLayout>
   ```

   Nazwa tego znacznika rozpoczyna się małą literą, co oznacza, że jest to właściwość obiektu `Application`.

   Obszar paska sterowania (ang. *control bar*) jest charakterystycznym obszarem kontenera. W tworzonej aplikacji umieścimy w nim logo oraz niektóre przyciski nawigacyjne.

5. Wewnątrz elementu `controlBarLayout` umieść samozamykający się znacznik `<s:BasicLayout/>`.

   ```
   <s:controlBarLayout>
   <s:BasicLayout/>
   </s:controlBarLayout>
   ```

   Samozamykające się znaczniki zastępują parę znaczników (otwierający i zamykający) zastosowaną np. w elemencie `controlBarLayout`. Zamknięcie sygnalizuje się prawym ukośnikiem umieszczonym przed znakiem większości kończącym definicję znacznika (`/>`).

   Dodanie znacznika `<s:BasicLayout>` oznacza, że w pasku sterowania aplikacji ma zostać zastosowane pozycjonowanie bezwzględne. Innymi słowy, każdy element umieszczony w tym obszarze musi mieć określone współrzędne *x* i *y*.

6. Poniżej elementu `controlBarLayout` dodaj nową parę znaczników `<s:controlBarContent>`.

   Wewnątrz tego znacznika zdefiniujesz elementy, które mają się pojawić w pasku sterowania.

7. Wewnątrz elementu `controlBarContent` umieść znacznik `Button` i jego właściwości `label` przypisz tekst **`Flex Grocer`**.

```
<s:Button label="Flex Grocer"/>
```

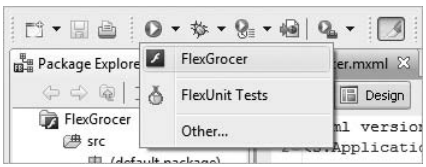
   Ustawienie właściwości `label` spowoduje wyświetlenie tekstu na przycisku. Ponieważ przycisk został dodany wewnątrz elementu `controlBarContent`, zostanie wyświetlony w obszarze paska sterowania aplikacji.

   Zanim przejdziesz dalej, sprawdź, czy wpisany kod wygląda tak jak poniższy.

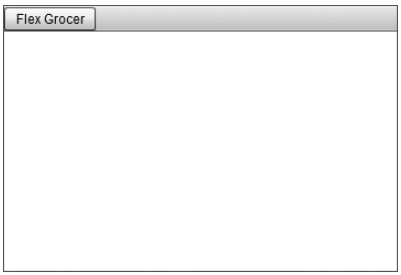
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="955" minHeight="600">
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>

   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button label="Flex Grocer"/>
   </s:controlBarContent>

</s:Application>
```

8. Po zweryfikowaniu kodu zapisz plik *FlexGrocer.mxml*. Sprawdź, czy w widoku *Problems* nie pojawiły się żadne komunikaty o błędach.

9. Z menu przycisku *Run* (uruchom) wybierz pozycję *FlexGrocer*, aby uruchomić aplikację w przeglądarce internetowej.



Po uruchomieniu aplikacji na górze ekranu powinien się pojawić szary obszar paska sterującego. Wewnątrz paska znajduje się przycisk *Flex Grocer*. Co prawda aplikacja nie jest jeszcze zbyt funkcjonalna, ale zdefiniowałeś już kilka właściwości, skorzystałeś z obiektu layoutu i dodałeś do kontenera obiekt potomny. Kolejne zadania będą łatwiejsze, a ich realizacja szybsza. Kiedy już skończysz podziwiać swoje dzieło, zamknij okno przeglądarki i przygotuj się do pracy w trybie projektowania.



Praca nad layoutem w trybie projektowania

Zdefiniowałeś już fragment layoutu aplikacji, korzystając z języka MXML. Teraz, by dodać kilka elementów i przypisać im właściwości, posłużysz się trybem projektowania.

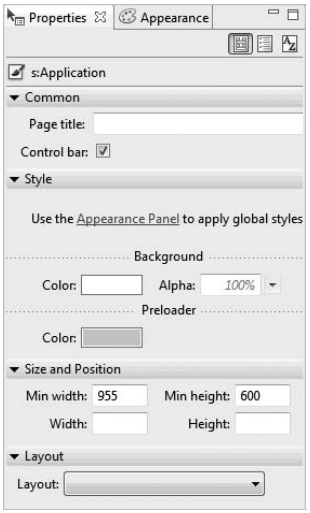
1. Włącz tryb projektowania (przycisk *Design*).

   Do przełączania między trybem projektowania i trybem edycji źródła używaj przycisków umieszczonych na pasku tytułowym w górnej części okna. Po włączeniu trybu projektowania zobaczysz podgląd opracowanego okna aplikacji.

2. Kliknij dowolne miejsce białego obszaru tła.

   W panelu *Properties* (właściwości) znajdującym się z prawej strony ekranu powinny się pojawić informacje o elemencie *s:Application*. W trybie projektowania panel ten będzie pomocny podczas ustawiania wartości poszczególnych właściwości komponentów.

Jeżeli panel Properties *nie jest wyświetlany, z menu* Window/Perspective *wybierz polecenie* Reset Perspective *(przywróć perspektywę). Spowoduje to przywrócenie domyślnej konfiguracji perspektywy dla trybu projektowania, więc panel* Properties *stanie się widoczny.*



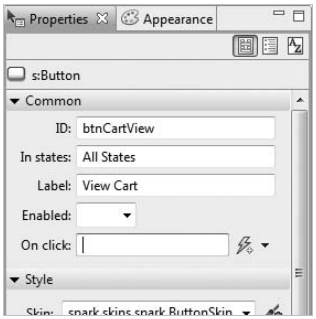
3. Kliknij przycisk *Flex Grocer*, który dodałeś w poprzednim ćwiczeniu.

   Po kliknięciu przycisku w panelu *Properties* zostaną wyświetlone właściwości elementu *s:Button*, więc możesz przystąpić do wprowadzania zmian dla wybranego przycisku.

4. Na dole panelu *Properties* znajduje się zwinięta kategoria *Size and Position* (rozmiar i położenie). Rozwiń ją, klikając ikonę trójkąta umieszczoną obok słowa *Size* (możliwe, że będziesz musiał przewinąć widok w dół, aby zobaczyć całą kategorię). Widoczne stały się pola edycyjne *Width* (szerokość), *Height* (wysokość), *X* oraz *Y*. W polach *X* i *Y* wpisz wartość **5**, definiując w ten sposób współrzędne przycisku równe (5, 5).

   Po zmianie współrzędnej *y* pasek sterujący zwiększył swoją wysokość, dopasowując ją do zaktualizowanej zawartości. W dalszej części książki zastosujesz style definiujące kolory i rozmiar logo. Na razie w miejscu logo pozostanie przycisk. Powyżej został przedstawiony sposób pozycjonowania oparty na bezwzględnych współrzędnych.

5. Odszukaj na ekranie widok *Components* (komponenty). Powinien znajdować się w lewym dolnym rogu okna Flash Buildera. Kliknij ikonę trójkąta przy nazwie *Controls* (kontrolki), by otworzyć katalog kontrolek, i przeciągnij przycisk (*Button*) do paska sterującego, umieszczając go blisko prawej krawędzi. W panelu *Properties* nadaj kontrolce *ID* równy **`btnCartView`** i etykietę **`View Cart`**.



Podczas wstawiania kontrolki pojawi się niebieska linia wskazująca wzajemne położenie (w pionie i poziomie) pozostałych komponentów. Pomaga to w szybkim umieszczaniu wielu komponentów w jednym rzędzie lub kolumnie.

   Na razie nie przejmuj się dokładnym umiejscowieniem kontrolek. W dalszej części tej lekcji nauczysz się posługiwać układem opartym na więzach w celu umieszczenia przycisku tak, by znajdował zawsze w odległości 10 pikseli od prawej krawędzi okna aplikacji.

6. Przeciągnij kolejny przycisk do paska sterującego, umieszczając go z lewej strony pierwszego. W panelu *Properties* nadaj kontrolce identyfikator **`btnCheckout`** i etykietę **`Checkout`**.

   Użytkownicy będą mogli nacisnąć ten przycisk, by skończyć zakupy i zamówić wybrane produkty. Tak jak poprzednio, dokładnym umiejscowieniem kontrolki zajmiesz się później, gdy nauczysz się korzystać z układu opartego na więzach.

7. Przeciągnij kontrolkę *Label* (etykieta) z katalogu *Controls* i umieść ją w prawym dolnym rogu ekranu. Kliknij dwukrotnie etykietę i w jej polu `text` wpisz **`(c) 2009, FlexGrocer`**.

Jak w przypadku przycisków: nie przejmuj się dokładnym umiejscowieniem etykiety — zajmiesz się tym później.

8. W panelu *Components* zwiń folder *Controls* i rozwiń *Layout*.

9. Przeciągnij kontener *Group* z katalogu *Layout* i umieść go w dużym białym obszarze poniżej paska sterowania. W panelu *Properties* ustaw *ID* kontenera na **`bodyGroup`**, wartości wysokości (pole *Height*) oraz szerokości (pole *Width*) na **`100%`**, a współrzędne *X* i *Y* na **`0`**.

10. Przy zaznaczonym kontenerze `bodyGroup` przewiń zawartość panelu *Properties* w dół. Powinno się pokazać rozwijane menu *Layout*. Rozwiń je i wybierz pozycję *spark.layouts.HorizontalLayout*, dzięki czemu kontener będzie układał obiekty potomne w poziomie.

    Dodany kontener `bodyGroup` będzie zawierał informacje o produktach oraz koszyk na zakupy. Pamiętaj o tym, że w kontenerze `Group` zdefiniowanym jako `HorizontalLayout` elementy są układane poziomo. W ten sposób informacje o produktach będą wyświetlane z lewej strony okna, a o koszyku — z prawej.

11. Przeciągnij kolejny kontener *Group* z katalogu *Layout* w panelu *Components* i umieść go wewnątrz kontenera `bodyGroup`. W panelu *Properties* ustaw *ID* kontenera na **`products`**, ustaw wysokość równą **`150`** i szerokość **`100%`**.

12. Skorzystaj z rozwijanego menu *Layout* znajdującego się na dole panelu *Properties* i wybierz pozycję *spark.layouts.VerticalLayout*, dzięki czemu elementy w tym kontenerze będą układane w pionie.

    Ten pionowy kontener będzie zawierał szczegółowe informacje o produkcie.

13. Zanim przystąpisz do realizacji kolejnego ćwiczenia, przełącz tryb pracy na edycję źródła i sprawdź, czy wygenerowany kod jest podobny do przedstawionego poniżej. Jeżeli jakiegoś znacznika brakuje lub jest błędnie zapisany, popraw to, korzystając z poniższego kodu. Oczywiście mogą wystąpić drobne różnice w wartościach współrzędnych `x` i `y` kontrolek, jeśli ustawiłeś je w trochę innych miejscach.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="955" minHeight="600">
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>

   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button id="btnCheckout" x="463" y="10" label="Checkout"/>
      <s:Button id="btnCartView" x="541" y="10" label="View Cart"/>
      <s:Button label="Flex Grocer" x="5" y="5"/>
   </s:controlBarContent>
   <s:Label x="518" y="320" text="(c) 2009, FlexGrocer"/>
   <s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
```

```
            <s:HorizontalLayout/>
         </s:layout>
         <s:Group width="100%" height="150" id="products">
            <s:layout>
               <s:VerticalLayout/>
            </s:layout>
         </s:Group>
      </s:Group>
   </s:Application>
```

Definiowanie sekcji produktów

Po upewnieniu się, że kod aplikacji odpowiada przedstawionemu powyżej, przełącz tryb pracy na projektowanie, ponieważ skorzystasz z niego podczas definiowania sekcji wyświetlającej produkty. Dodasz teraz kontrolki odpowiadające wszystkim produktom dostępnym w projektowanym sklepie internetowym.

Czasem podczas przełączania między trybami edycji kodu i projektowania znika panel Properties*, który powinien być widoczny w trybie projektowania. Jeżeli panel zniknął, z menu* Window *wybierz pozycję* Properties.

1. Przeciągnij kontrolkę *Label* z folderu *Controls* w widoku *Components* i umieść ją w pionowym kontenerze `products`, który dodałeś w poprzednim ćwiczeniu. W trybie projektowania wspomniany pionowy kontener można rozpoznać po delikatnej ramce rozpoczynającej się tuż poniżej paska sterującego, a kończącej 150 pikseli niżej. Etykietę możesz umieścić w dowolnym miejscu tego obszaru.
2. Identyfikatorowi (*ID*) kontrolki przypisz nazwę **`prodName`**, a właściwość *text* ustaw na **`Milk`**.
3. Przeciągnij drugą kontrolkę *Label* na miejsce poniżej pierwszej. Wpisz jej *ID* — **`price`**, oraz *text* — **`$1.99`**.

   Ponieważ dodane etykiety są umieszczone w kontenerze `Group` ustawionym jako `Vertical` ↪`Layout`, nazwa produktu jest wyświetlana nad ceną.

Jeżeli otworzysz widok Outline*, klikając zakładkę* Outline *(znajdującą się obok zakładki* Components*, z której już korzystałeś), będziesz mógł zobaczyć hierarchię elementów tworzących aplikację. Głównym elementem jest znacznik* `<s:Application>`*, w którym są zawarte elementy potomne: etykieta (*`Label`*) i kontener* `bodyGroup`*, a także właściwości* `controlBarContent` *i* `controlBarLayout`*. Widoczne są również inne składniki elementów* `controlBarContent` *i* `bodyGroup`*. Jeżeli rozwiniesz gałąź kontenera* `Group` *o nazwie* `products`*, zobaczysz dwie ostatnio dodane etykiety. Widok* Outline *jest bardzo użyteczny, gdy chcesz dokonać zmian któregoś z komponentów. Wybranie np. kontenera* `products` *może być trudne w trybie projektowania. Dużo łatwiejsze jest wybranie go kliknięciem w widoku* `Outline`*.*

4. Dodaj kontrolkę *Button* poniżej dwóch etykiet, nadaj jej identyfikator **`add`** oraz etykietę **`AddToCart`**.

Przy każdym produkcie powinna być wyświetlana nazwa i cena. Przycisk *AddToCart* umożliwia użytkownikom umieszczenie wybranych produktów w koszykach na zakupy. Ponieważ dwie kontrolki *Label* oraz kontrolka *Button* są umieszczone w kontenerze z pionowym ułożeniem elementów, są wyświetlane jedna nad drugą. Funkcjonalnością przycisku zajmiesz się w kolejnej lekcji.

5. Zapisz plik i kliknij przycisk *Run*.

   W uruchomionej aplikacji widać wyraźnie różnicę pomiędzy elementami umieszczonymi w pasku sterującym a tymi, które znajdują się w głównym obszarze aplikacji.

Praca z layoutami opartymi na więzach

Flex obsługuje layouty oparte na więzach, co pozwala na dowolne rozmieszczanie elementów interfejsu użytkownika przy zachowaniu dużej dokładności pozycjonowania. Według więzów nałożonych na elementy jest ustalana wielkość i położenie tych elementów, kiedy użytkownik zmienia wielkość okna aplikacji. Taka metoda ustalania rozmiaru i położenia elementów różni się od tej, w której stosowane są zagnieżdżone kontenery (takie jak np. kontenery `Group` wykorzystane w poprzednim ćwiczeniu).

W przypadku layoutów opartych na więzach wszystkie kontrolki są rozmieszczane względem krawędzi nadrzędnego kontenera wykorzystującego obiekt `BasicLayout` pozwalający na pozycjonowanie bezwzględne. Obiektu tego można użyć z dowolnym kontenerem typu `Group` i `SkinnableContainer`, włączając w to `Application` i `Panel`. Wyjątek stanowią specjalizowane kontenery, na przykład `Form` (które poznasz w kolejnych lekcjach).

Kontenery korzystające z obiektu `BasicLayout` wymagają co prawda określenia bezwzględnych współrzędnych elementów, jednak więzy umożliwiają dynamiczne dopasowanie przygotowanego w ten sposób layoutu do wielkości okna aplikacji wyświetlanej w przeglądarce internetowej. Jeśli na przykład chcesz,

by etykieta była zawsze wyświetlana w prawym dolnym rogu okna aplikacji (bez względu na rozmiar okna przeglądarki), możesz zakotwiczyć kontrolkę do prawej krawędzi nadrzędnego kontenera. Oznacza to, że położenie kontrolki będzie ustalane względem prawej krawędzi kontenera.

Zakotwiczenia layoutu używane są do ustalenia, w jaki sposób kontrolka ma być wyświetlana względem krawędzi nadrzędnego kontenera. Aby sprawić, by znajdowała się ona w wyznaczonej odległości od prawej i dolnej krawędzi kontenera, należy zaznaczyć pola wyboru umieszczone poniżej i z prawej strony kontrolki w obszarze *Constraints* sekcji *Layout* w panelu *Properties*. W polach tekstowych podawana jest odległość (w liczbie pikseli), o jaką krawędzie zakotwiczonej kontrolki mają być oddalone od brzegów kontenera.

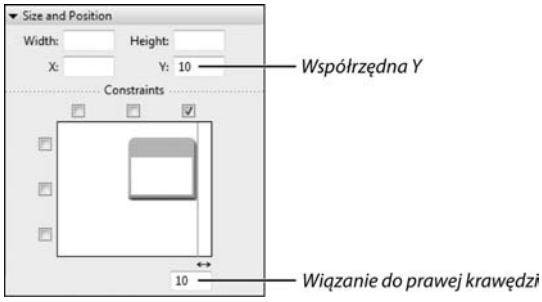
Flex udostępnia więzy do górnej, dolnej, lewej i prawej krawędzi kontenera nadrzędnego oraz do jego środka w poziomie i w pionie.

Wszystkie więzy ustawiane są względem krawędzi kontenera, jeżeli zastosowano w nim pozycjonowanie bezwzględne (obiekt `BasicLayout`*), więc nie mogą być ustawiane względem innych kontrolek lub kontenerów.*

1. Otwórz plik *FlexGrocer.mxml*, używany w poprzedniej lekcji.

   Jeżeli nie ukończyłeś poprzedniej lekcji lub przygotowany kod nie działa prawidłowo, możesz zaimportować projekt *FlexGrocer.fxp* z katalogu *Lekcja03/intermediate*. Szczegółowe informacje na temat importowania projektu znajdziesz w dodatku A. Skorzystaj z uwag tam zawartych za każdym razem, gdy opuścisz którąś lekcję lub natrafisz na problem, którego nie umiesz rozwiązać.

2. Znajdź i zaznacz przycisk *Checkout*. W obszarze *Constraints* sekcji *Size and Position* panelu *Properties* zdefiniuj wiązanie prawej krawędzi przycisku do prawej krawędzi kontenera i wpisz odległość równą **10** pikseli. Sprawdź, czy w polu *Y* jest ustawiona wartość **10**.



   W celu uwiązania przycisku do prawej krawędzi kliknij pole wyboru umieszczone najbardziej z prawej strony ponad ikoną przycisku w obszarze *Constraints*. W wyświetlonym polu tekstowym wpisz liczbę pikseli, o jaką przycisk ma być oddalony od brzegu kontenera. Jeśli etykieta zniknie z ekranu, skorzystaj z pasków przewijania podglądu w trybie projektowania. Zwykle wyświetlany jest tylko fragment okna aplikacji, więc czasem — by zobaczyć pewien element — konieczne jest przewinięcie zawartości.

3. Znajdź i zaznacz przycisk *View Cart*. Dodaj więzy tak, aby prawa krawędź przycisku była oddalona o **90** pikseli od prawej krawędzi kontenera. Sprawdź, czy w polu *Y* jest ustawiona wartość **10**.

   Po wprowadzeniu zmian oba przyciski nawigacyjne zawsze będą zakotwiczone do górnego prawego rogu kontenera, niezależnie od rozmiaru okna przeglądarki.

4. Znajdź i zaznacz etykietę zawierającą notkę o prawach autorskich. Ustaw więzy etykiety tak, aby była ona oddalona o 10 pikseli od dolnej i prawej krawędzi zawierającego ją kontenera. Kliknij pole wyboru w prawym górnym rogu obszaru *Constraints* i wpisz **10** w wyświetlonym polu tekstowym. Następnie kliknij dolne pole wyboru i również wpisz wartość **10**.

   Ponieważ etykieta znajduje się poniżej innych kontenerów, prawdopodobnie najprościej będzie wskazać ją w widoku *Outline*. Wprowadzone ustawienia gwarantują, że niezależnie od szerokości kontrolki *Label* jej prawy dolny róg będzie zawsze umieszczony w pozycji 10 pikseli powyżej i 10 pikseli w lewą stronę od prawego dolnego narożnika aplikacji.

   Jeżeli teraz przełączysz program w tryb edycji źródła (*Source*), kod powinien przypominać ten zamieszczony na poniższym listingu.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="955" minHeight="600">
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>

   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button id="btnCheckout" label="Checkout" right="10" y="10"/>
      <s:Button id="btnCartView" label="View Cart" right="90" y="10"/>
      <s:Button label="Flex Grocer" x="5" y="5"/>
   </s:controlBarContent>
   <s:Label text="(c) 2009, FlexGrocer" right="10" bottom="10"/>
   <s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
         <s:HorizontalLayout/>
      </s:layout>
      <s:Group width="100%" height="150" id="products">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
         <s:Button label="AddToCart" id="add"/>
      </s:Group>
   </s:Group>
</s:Application>
```

W Twoim kodzie mogą wystąpić niewielkie różnice spowodowane kolejnością, w jakiej dodawałeś elementy i definiowałeś ich właściwości. Nie musisz się tym przejmować, ponieważ w tym przypadku kolejność nie jest istotna. Każdy kontener i każda kontrolka dodawane

w trybie projektowania są reprezentowane w kodzie programu przez znacznik. Elementy dodawane do kontenera są reprezentowane przez znaczniki potomne znacznika kontenera. Zauważ także, że więzy układu są oznaczane jako atrybuty odpowiedniego kontenera.

5. Przełącz program z powrotem w tryb projektowania i wewnątrz kontenera `bodyGroup` umieść drugi kontener *Group* (element `bodyGroup` jest pierwszym, który dodałeś; jego wysokość i szerokość są ustawione na 100%). W polu *ID* wpisz **`cartGroup`**, wyczyść pole szerokości (*Width*), a w polu wysokości (*Height*) wpisz **`100%`**. Pamiętaj, że w przypadku problemów z odszukaniem komponentów w trybie projektowania zawsze możesz skorzystać z widoku *Outline*.

   Jeśli przez przypadek umieściłeś element `Group` w niewłaściwym miejscu, najprostszym sposobem naprawienia błędu jest przełączenie się do trybu edycji źródła i ręczne przeniesienie znaczników. Na poniższym listingu został przedstawiony omawiany fragment kodu.

```
<s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
         <s:HorizontalLayout/>
      </s:layout>
      <s:Group width="100%" height="150" id="products">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
         <s:Button label="AddToCart" id="add"/>
      </s:Group>
      <s:Group height="100%" id="cartGroup">
      </s:Group>
   </s:Group>
```

6. W trybie projektowania ustaw typ layoutu kontenera `cartGroup` na *VerticalLayout*.

   Jeżeli nie umiesz znaleźć kontenera `cartGroup`, zaznacz go w widoku *Outline* i przewiń okno podglądu trybu projektowania, by zobaczyć podświetlony kontener.

7. Dodaj kontrolkę *Label* do kontenera `cartGroup` i w polu właściwości *text* wpisz **`Your Cart Total: $`**.

   Po prawej stronie wyświetlanych informacji o produktach będzie zawsze widoczna ogólna informacja o koszyku, wskazująca, czy zawiera on jakieś produkty i jaki jest aktualny koszt zakupów.

Szerokość kontenera `products` została ustawiona na 100%, czyli tak, aby zajmował całą dostępną przestrzeń. Później jednak z jego prawej strony dodałeś kontener `cartGroup` i etykietę. Czy to oznacza, że wykorzystywane jest ponad 100% szerokości? We Fleksie szerokość i wysokość kontenera może przekraczać 100%. Przy ustalaniu rozmiarów i rozmieszczenia elementów dostępna przestrzeń jest dzielona proporcjonalnie do przypisanych szerokości i wysokości. Ponieważ w omawianym przypadku żądana szerokość przekracza maksymalną, każdej kontrolce zostanie przydzielona szerokość, jaka wynika z podziału dostępnego miejsca. Jeśli wśród kontrolek znajdują się takie, których szerokości są ustawione na stałe (czyli zastosowano wielkości wyrażone w pikselach, a nie procentach), wartości te zostaną odjęte od całkowitej dostępnej szerokości, a dopiero później dojdzie do jej podziału na pozostałe elementy.

8. Przeciągnij kontrolkę *Button* z katalogu *Controls* w widoku *Components* i umieść ją poniżej ostatnio wprowadzonej etykiety. W polu właściwości *label* wpisz **`View Cart`**.

   Przycisk ten będzie służył do wyświetlania pełnej zawartości koszyka.

   Jeśli przez przypadek umieściłeś dodawane komponenty w niewłaściwym miejscu, najprostszym sposobem naprawienia błędu jest przełączenie się do trybu edycji źródła i ręczne przeniesienie znaczników. Na poniższym listingu został przedstawiony omawiany fragment kodu.

```
<s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
         <s:HorizontalLayout/>
      </s:layout>
      <s:Group width="100%" height="150" id="products">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
         <s:Button label="AddToCart" id="add"/>
      </s:Group>
      <s:Group height="100%" id="cartGroup">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Your Cart Total: $"/>
         <s:Button label="View Cart"/>
      </s:Group>
   </s:Group>
```

9. W widoku *Outline* zaznacz znacznik `Application`. W panelu *Properties* wyczyść pola *Min width* (minimalna szerokość) i *Min height* (minimalna wysokość).

   Po uruchomieniu aplikacji i kilkakrotnej zmianie rozmiaru okna przeglądarki będzie można się przekonać, że przyciski i etykiety są prawidłowo rozmieszczane. Wartości minimalnej szerokości i wysokości mogłyby uniemożliwić sprawdzenie tego zachowania w przypadku mniejszych ekranów.

10. Zapisz plik i uruchom aplikację poleceniem *Run*.

Praca ze stanami widoku

Flex umożliwia projektowanie aplikacji zmieniających wygląd w zależności od działań użytkownika. Na przykład aplikacja sklepu internetowego rozpoczyna działanie od wyświetlenia użytkownikowi informacji o produktach, które może kupić. Gdy użytkownik zaczyna dodawać produkty do koszyka, widok powinien być zaktualizowany tak, by klient miał kontrolę nad zawartością koszyka. Może być na przykład wyświetlany całkowity koszt wybranych produktów. Użytkownik powinien mieć także możliwość przeglądania zawartości koszyka i zarządzania nią.

Tworzenie stanów widoku

Dodawanie tego typu mechanizmu we Fleksie realizuje się za pośrednictwem stanów widoku (ang. *view states*). Stan widoku jest jednym z wielu sposobów wyświetlania definiowanych dla całej aplikacji bądź jej elementów. Każdy plik MXML posiada co najmniej jedno ustawienie nazywane *podstawowym stanem* (ang. *default state*), które odzwierciedla domyślny wygląd aplikacji.

Dodatkowe stany w MXML-u to zmodyfikowane ustawienia widoku podstawowego lub innych stanów.

1. Otwórz plik *FlexGrocer.mxml*, w którym pracowałeś w poprzednim ćwiczeniu.

   Jeżeli nie ukończyłeś poprzedniej lekcji lub przygotowany kod nie działa prawidłowo, możesz zaimportować projekt *FlexGrocer-PreStates.fxp* z katalogu *Lekcja03/intermediate*. Szczegółowe informacje na temat importowania projektu znajdziesz w dodatku A. Skorzystaj z uwag tam zawartych za każdym razem, gdy opuścisz którąś lekcję lub natrafisz na problem, którego nie umiesz rozwiązać.

2. Przejdź do trybu projektowania. Jeżeli widok *States* (stany) nie jest widoczny, z menu *Window* wybierz pozycję *States*.

   W panelu *States* powinien być widoczny jeden stan odpowiadający domyślnemu layoutowi aplikacji.

3. Utwórz nowy stan o nazwie **`cartView`**, oparty na stanie `<State1>`.

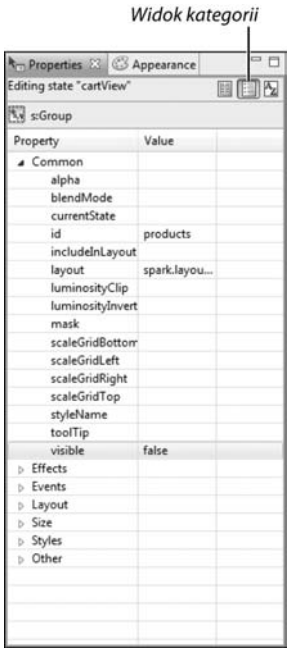
   Nowy stan tworzy się przez kliknięcie przycisku *New State* u góry panelu *States* lub przez kliknięcie prawym przyciskiem myszy w panelu i wybranie z podręcznego menu opcji *New*. W widoku `cartView` będą wyświetlane szczegółowe informacje o wszystkich produktach umieszczonych przez użytkownika w koszyku.

4. Po zaznaczeniu na liście stanów pozycji `cartView` kliknij kontener `products` i nadaj jego szerokości i wysokości wartości **0**. Następnie zaznacz kontener `cartGroup` i ustaw jego szerokość oraz wysokość na **100%**.

   W stanie widoku `cartView` zawartość koszyka ma całkowicie zastąpić informacje o produktach wyświetlane na środku ekranu. Z tego powodu zmieniłeś rozmiar kontenera produktów tak, aby w ogóle nie zajmował miejsca, a rozmiar kontenera `cartGroup` tak, aby zajmował całą dostępną przestrzeń.

   Po wprowadzeniu opisanych zmian na ekranie zrobił się niemały bałagan, ponieważ kontrolki pozachodziły na siebie. To bardzo ważna lekcja — we Fleksie właściwości wysokości i szerokości służą do wyznaczenia położenia poszczególnych elementów na ekranie. W tym przypadku kontener `products` przestał zajmować miejsce, więc kontener `cartGroup` przesunął się w lewo, zajmując dostępną przestrzeń. Trzeba jednak wiedzieć, że ustawienie rozmiaru elementu w taki sposób, by nie zajmował miejsca na ekranie, nie oznacza tego samego, co wyłączenie widoczności.

5. Zaznacz kontener `products` i zmień właściwość `visible` (widoczny) na **`false`** (fałsz). Aby to zrobić, w panelu *Properties* wciśnij przycisk *Category View* (widok kategorii), odszukaj właściwość *visible* i zmień jej wartość na ***false***.



Co prawda było to już wielokrotnie powtarzane w tej lekcji, ale jest bardzo istotne, więc warte powtarzania: jeżeli masz problem z odszukaniem kontenera w trybie podglądu okna aplikacji, skorzystaj z widoku Outline.

6. W widoku *States* zaznacz stan `cartView`, a następnie z katalogu *Controls* w panelu *Components* przeciągnij kontrolkę *DataGrid* (siatka danych) i umieść ją poniżej przycisku *View Cart*. Nadaj kontrolce identyfikator `ID` równy **`dgCart`** i ustaw jej szerokość na **`100%`**.

   W jednej z następnych lekcji kontrolka *DataGrid* zostanie użyta do wyświetlenia całej zawartości koszyka.

   Upewnij się, że umieściłeś kontrolkę *DataGrid* w kontenerze *cartGroup*. Jeżeli przypadkowo umieścisz kontrolkę poza kontenerem, kod źródłowy aplikacji będzie wyglądał trochę inaczej od zamieszczonego poniżej.

   Włącz tryb edycji źródła. Na poniższym listingu został zamieszczony pełny kod aplikacji (pogrubioną czcionką są zaznaczone fragmenty dodane w tym ćwiczeniu).

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx">
   <s:states>
      <s:State name="State1"/>
      <s:State name="cartView"/>
   </s:states>
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>
   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button id="btnCheckout" label="Checkout" right="10" y="10"/>
      <s:Button id="btnCartView" label="View Cart" right="90" y="10"/>
      <s:Button label="Flex Grocer" x="5" y="5"/>
   </s:controlBarContent>
   <s:Label text="(c) 2009, FlexGrocer" right="10" bottom="10"/>
   <s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
         <s:HorizontalLayout/>
      </s:layout>
      <s:Group width="100%" height="150" id="products"
           width.cartView="0" height.cartView="0"
           visible.cartView="false">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
         <s:Button label="AddToCart" id="add"/>
      </s:Group>
      <s:Group height="100%" id="cartGroup" width.cartView="100%">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Your Cart Total: $"/>
         <s:Button label="View Cart"/>
         <mx:DataGrid includeIn="cartView" id="dgCart" width="100%">
```

```
                <mx:columns>
                    <mx:DataGridColumn headerText="Column 1"
                        dataField="col1"/>
                    <mx:DataGridColumn headerText="Column 2"
                        dataField="col2"/>
                    <mx:DataGridColumn headerText="Column 3"
                        dataField="col3"/>
                </mx:columns>
            </mx:DataGrid>
        </s:Group>
    </s:Group>
</s:Application>
```

7. Zapisz plik.

   Zwróć uwagę na nowe elementy i konstrukcje, które pojawiły się w kodzie. Po pierwsze, w klasie `DataGrid` została wykorzystana właściwość `includeIn`, która odpowiada za wyświetlenie kontrolki tylko we wskazanym stanie (`cartView`). Po drugie, kontener `products` nadal ma szerokość 100% i wysokość 150 pikseli, jednak zostały dodane dwie właściwości: `width.cartView="0"` i `height.cartView="0"`. Taka konstrukcja powoduje ustawienie właściwości tylko w stanie `cartView`.

W działaniu uruchomionego testowo programu nie powinny wystąpić żadne widoczne różnice, ponieważ nie zostały jeszcze dodane możliwości przełączania się przez użytkownika między stanami widoku. Funkcje nawigacyjne zostaną dodane w następnym ćwiczeniu.

Sterowanie stanami widoku

W MXML-u każdy element ma właściwość `currentState`. Można jej użyć do wybrania stanu wyświetlanego w danej sytuacji.

1. Otwórz plik *FlexGrocer.mxml*, w którym pracowałeś w poprzednim ćwiczeniu.

   Jeżeli nie ukończyłeś poprzedniej lekcji lub przygotowany kod nie działa prawidłowo, możesz zaimportować projekt *FlexGrocer-PreControl.fxp* z katalogu *Lekcja03/intermediate*. Szczegółowe informacje na temat importowania projektu znajdziesz w dodatku A. Skorzystaj z uwag tam zawartych za każdym razem, gdy opuścisz którąś lekcję lub natrafisz na problem, którego nie umiesz rozwiązać.

2. Włącz tryb projektowania, przejdź do widoku *States* (jeżeli jest niewidoczny, włącz go) i zaznacz stan *State1*.

   Do bazowego stanu widoku dodasz możliwość przejścia do innych stanów aplikacji.

3. Zaznacz przycisk `View Cart` umieszczony w kontenerze `cartGroup`. W panelu *Properties* w polu *On click* wpisz `this.currentState='cartView'`.

   Zdarzenia takie jak kliknięcie przycisku zostaną szczegółowo omówione w lekcji 5., „Obsługa zdarzeń”. W tej chwili wystarczy, żebyś wiedział, że po kliknięciu przycisku widok zmieni się na stan `cartView`.

W nazwach stanów widoku rozróżniane są wielkie i małe litery, dlatego musisz wpisać taką samą nazwę, jak w poprzednim ćwiczeniu. Wpisując nazwę stanu widoku w trybie projektowania, musisz ją ująć w apostrofy.

4. Zaznacz przycisk *View Cart* umieszczony w pasku sterującym. W panelu *Properties* w polu *On click* również wpisz `this.currentState='cartView'`. Dzięki temu do stanu cartView można się będzie przełączyć na dwa sposoby.

5. Przełącz stan widoku na *cartView*. Dodaj nową kontrolkę *Button* poniżej kontrolki *DataGrid*, nadając jej etykietę **`Continue Shopping`** (kontynuuj zakupy) oraz ustawiając wartość właściwości `click` na **`this.currentState=''`**.

   Przypisanie do właściwości `currentState` pustego ciągu spowoduje powrót do bazowego stanu widoku aplikacji.

6. Z widoku `cartView` usuń przycisk *View Cart* znajdujący się w kontenerze `cartGroup`.

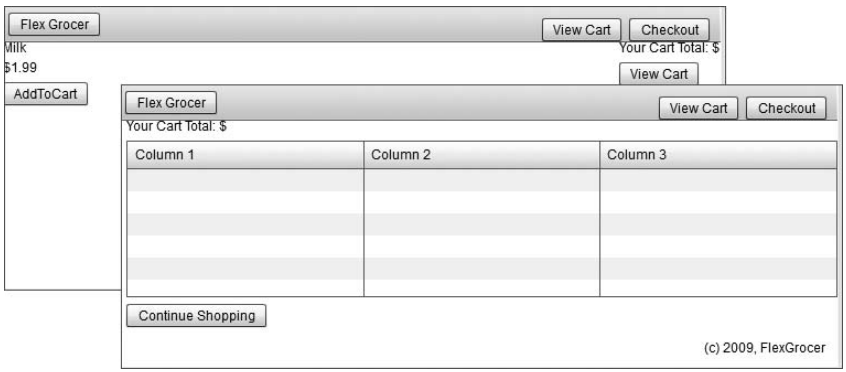
   Gdy użytkownik przegląda zawartość koszyka, nie jest mu potrzebny przycisk *View Cart*. W trybie projektowania przycisk możesz usunąć, zaznaczając go i wciskając klawisz *Delete*.

   Pełny kod aplikacji został przedstawiony na poniższym listingu.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx">
   <s:states>
      <s:State name="State1"/>
      <s:State name="cartView"/>
   </s:states>
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>
   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button id="btnCheckout" label="Checkout" right="10" y="10"/>
      <s:Button id="btnCartView" label="View Cart" right="90" y="10"
         click.State1="this.currentState='cartView'"/>
      <s:Button label="Flex Grocer" x="5" y="5"/>
   </s:controlBarContent>
   <s:Label text="(c) 2009, FlexGrocer" right="10" bottom="10"/>
   <s:Group x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:layout>
         <s:HorizontalLayout/>
      </s:layout>
      <s:Group width="100%" height="150" id="products"
           width.cartView="0" height.cartView="0"
           visible.cartView="false">
         <s:layout>
            <s:VerticalLayout/>
         </s:layout>
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
```

```
            <s:Button label="AddToCart" id="add"/>
         </s:Group>
         <s:Group height="100%" id="cartGroup" width.cartView="100%">
            <s:layout>
               <s:VerticalLayout/>
            </s:layout>
            <s:Label text="Your Cart Total: $"/>
            <s:Button label="View Cart" click="this.currentState='cartView'"
               includeIn="State1"/>
            <mx:DataGrid includeIn="cartView" id="dgCart" width="100%">
               <mx:columns>
                  <mx:DataGridColumn headerText="Column 1"
                     dataField="col1"/>
                  <mx:DataGridColumn headerText="Column 2"
                     dataField="col2"/>
                  <mx:DataGridColumn headerText="Column 3"
                     dataField="col3"/>
               </mx:columns>
            </mx:DataGrid>
            <s:Button includeIn="cartView" label="Continue Shopping"
               click="this.currentState=''"/>
         </s:Group>
      </s:Group>
   </s:Application>
```

7. Zapisz plik i przetestuj aplikację. Możesz teraz przełączać stany widoku, klikając przyciski, do których dodałeś kod.



Refaktoryzacja

Refaktoryzacja jest jednym z najmniej znanych, a zarazem jednym z najbardziej użytecznych narzędzi dostępnych dla programistów. Programiści Fleksa i języka ActionScript powinni je szczególnie docenić, ponieważ podczas tworzenia prototypów i kolejnych faz projektów zawierających dynamiczne interfejsy często dochodzi do zmian.

Refaktoryzacja jest procesem, w którym dochodzi do reorganizacji kodu bez ingerencji w jego funkcjonowanie, dzięki czemu łatwiej go utrzymywać i modyfikować w przyszłości. O aplikacji trzeba myśleć długofalowo — trzeba będzie ją poprawiać, zmieniać architekturę w celu dopasowania do nowych

możliwości oraz, co ważne, przygotować na zmianę zespołu programistów. Zawsze jednak musi obowiązywać zasada: po pomyślnym zakończeniu procesu refaktoryzacji użytkownik niemający wglądu do kodu nie powinien zauważyć żadnej zmiany w funkcjonowaniu aplikacji.

Wielu programistów uważa ten pogląd za niesłuszny, a samą refaktoryzację za zbędną. Po co spędzać dużo czasu na modyfikowaniu kodu, który się właśnie napisało, skoro nie przyniesie to żadnych zmian w funkcjonowaniu aplikacji? Na tak postawione pytanie można odpowiedzieć na wiele sposobów. Poniżej kilka możliwych argumentów.

- **Wartości edukacyjne**. Nauka nowego języka i późniejsze jego używanie wymaga ciągłego poszerzania wiedzy. Każdego dnia zdobywa się nowe umiejętności i poznaje nowe techniki. Prowadzi to często do konstatacji, że kod napisany kilka dni, tygodni lub miesięcy temu jest źle napisany, a może nawet nie działa prawidłowo w niektórych sytuacjach. Analizowanie napisanego wcześniej kodu połączone z chęcią wprowadzania zmian może pomóc w tworzeniu dużo lepszego kodu, bardziej spójnego i przygotowanego na modyfikacje.
- **Powielanie kodu**. Podczas programowania bardzo często zdarza się sytuacja, w której w wielu miejscach aplikacji wymagane jest przeprowadzenie tych samych operacji. Niestety z uwagi na pośpiech typowym rozwiązaniem jest powielanie fragmentów kodu. Najpoważniejszy problem wynikający z takiego postępowania pojawia się wtedy, gdy do kodu trzeba wprowadzić jakieś poprawki. Programista musi wprowadzić zmiany we wszystkich wystąpieniach powielonego kodu. Najistotniejsze jest unikanie za wszelką cenę powielania kodu. Dzięki temu zaoszczędzimy dużo czasu, a programowanie stanie się zdecydowanie bardziej efektywne.
- **Spojrzenie z szerszej perspektywy**. Niekiedy na początku pisania kodu trudno jest przewidzieć (o ile to w ogóle możliwe), jak będzie zbudowana cała aplikacja i jak jej elementy będą ze sobą współpracować. Jeżeli fragmenty kodu napisane wcześniej są „niemodyfikowalne", podczas próby połączenia ich z innymi elementami aplikacji może się okazać, że konieczne będzie zupełne przebudowanie fragmentów programu. Jeżeli będziesz regularnie refaktoryzował kod, możesz na bieżąco „szlifować" poszczególne elementy, tak aby pod koniec tworzenia aplikacji wszystko do siebie pasowało.

Mamy kilka powodów, dla których podejmujemy temat refaktoryzacji. Po pierwsze, wielu programistów Fleksa i języka ActionScript prezentuje dosyć „skostniałe" podejście do programowania, w którym nie ma miejsca na refaktoryzację. Z naszych obserwacji wynika, że tego typu programiści wolą walczyć z językiem, zamiast nauczyć się go efektywnie wykorzystywać. Chcemy po prostu ustrzec czytelników przed problemami.

Po drugie, czytając tę książkę i wykonując ćwiczenia, będziesz się uczyć. W wielu miejscach poznasz wiele rozwiązań prowadzących do tego samego celu. Czasami niemożliwe jest wskazanie jedynej słusznej metody, ponieważ zbyt wiele jest zależności. Kiedy znajdziesz nowy, lepszy sposób na rozwiązanie jakiegoś problemu, możesz na tym zakończyć proces refaktoryzacji. Wynikają z tego dwie korzyści: poznajesz wiele możliwości rozwiązania danego zadania (oraz wiesz, dlaczego w danej sytuacji zastosować takie, a nie inne rozwiązanie) oraz dopracowujesz kod aplikacji, który może się przydać w przyszłości.

Czas przejść do praktyki. Utworzoną aplikację poddasz refaktoryzacji, dzięki czemu łatwiej będzie nad nią pracować w kolejnych lekcjach.

Stosowanie złożonych kontenerów

W tej lekcji dowiedziałeś się, że w większości kontenerów można zastosować obiekty layoutu, które wpływają na sposób układania elementów potomnych. Dokonuje się tego poprzez dodanie obiektu klasy `LayoutObject` do kontenera `Group`, stosując właściwość `layout`, co zostało przedstawione na poniższym przykładzie:

```
<s:Group>
   <s:layout>
      <s:HorizontalLayout/>
   </s:layout>

   <s:Button label="1"/>
   <s:Button label="2"/>
   <s:Button label="3"/>
</s:Group>
```

Chociaż takie rozwiązanie jest najbardziej elastyczne, wymaga wprowadzenia dodatkowego kodu za każdym razem, gdy tworzymy kontener `Group`. W małych aplikacjach nie jest to poważnym problemem, jednak w znacznie większych projektach dodawanie obiektów layoutu do setek kontenerów może być uciążliwe. Flex wychodzi naprzeciw takim problemom, umożliwiając tworzenie nowych komponentów złożonych z innych obiektów i właściwości. Tak utworzone konstrukcje można później stosować w projekcie, ułatwiając sobie pracę.

Flex udostępnia dwa komponenty zbudowane właśnie w ten sposób: `VGroup` i `HGroup`. W poniższej tabeli zostało zamieszczone porównanie kodu wykorzystującego kontener `Group` oraz dwa opisane komponenty.

Alternatywna implementacja kontenerów

Wersja z właściwością `layout`	Wersja ze złożonymi kontenerami

`<s:Group>`
`  <s:layout>`
`  <s:HorizontalLayout/>`
`  </s:layout>`
 	
`  <s:Button label="1"/>`
`  <s:Button label="2"/>`
`  <s:Button label="3"/>`
`</s:Group>`
	`<s:HGroup>`
	`  <s:Button label="1"/>`
	`  <s:Button label="2"/>`
	`  <s:Button label="3"/>`
	`</s:HGroup>`

`<s:Group>`
`  <s:layout>`
`  <s:VerticalLayout/>`
`  </s:layout>`
 	
`  <s:Button label="1"/>`
`  <s:Button label="2"/>`
`  <s:Button label="3"/>`
`</s:Group>`
	`<s:VGroup>`
	`  <s:Button label="1"/>`
	`  <s:Button label="2"/>`
	`  <s:Button label="3"/>`
	`</s:VGroup>`

Jeśli zajrzysz do kodu źródłowego komponentów `HGroup` i `VGroup`, zauważysz, że są bardziej skomplikowane od zastosowanych w omawianej aplikacji. W lekcji 9. „Tworzenie komponentów” i lekcji 18. „Tworzenie komponentów w języku ActionScript” dowiesz się, jak tworzyć własne komponenty, i zobaczysz, jak wielokrotnie wykorzystać kod.

Refaktoryzacja kodu aplikacji

W tym ćwiczeniu zamienisz wszystkie kontenery `Group` z obiektami `HorizontalLayout` i `Vertical` ↪`Layout` na komponenty `HGroup` i `VGroup`. Celem tych działań jest zmodyfikowanie wewnętrznej struktury aplikacji bez wprowadzenia zmian w jej funkcjonowaniu i wyglądzie.

1. Otwórz plik *FlexGrocer.mxml*, w którym pracowałeś w poprzednim ćwiczeniu.

   Jeżeli nie ukończyłeś poprzedniej lekcji lub przygotowany kod nie działa prawidłowo, możesz zaimportować projekt *FlexGrocer-PreRefactor.fxp* z katalogu *Lekcja03/intermediate*. Szczegółowe informacje na temat importowania projektu znajdziesz w dodatku A. Skorzystaj z uwag tam zawartych za każdym razem, gdy opuścisz którąś lekcję lub natrafisz na problem, którego nie umiesz rozwiązać.

2. Włącz tryb edycji kodu źródłowego.

3. Odszukaj znacznik `Group` o identyfikatorze `bodyGroup` i zamień go na `HGroup`. Nie zapomnij zmienić również znacznika zamykającego.

4. Z elementu `bodyGroup` usuń właściwość `layout` i obiekt `HorizontalLayout`.

5. Odszukaj znacznik `Group` o identyfikatorze `products` i zamień go na `VGroup`. Nie zapomnij zmienić również znacznika zamykającego.

6. Z elementu `products` usuń właściwość `layout` i obiekt `VerticalLayout`.

7. Te same czynności powtórz dla grupy `cartGroup`.

   Po pomyślnym zakończeniu refaktoryzacji kod aplikacji powinien przypominać ten z poniższego listingu.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx">
   <s:states>
      <s:State name="State1"/>
      <s:State name="cartView"/>
   </s:states>
   <fx:Declarations>
      <!-- Place non-visual elements (e.g., services, value objects) here -->
   </fx:Declarations>
   <s:controlBarLayout>
      <s:BasicLayout/>
   </s:controlBarLayout>

   <s:controlBarContent>
      <s:Button id="btnCheckout" label="Checkout" right="10" y="10"/>
      <s:Button id="btnCartView" label="View Cart" right="90" y="10"
         click.State1="this.currentState='cartView'"/>
      <s:Button label="Flex Grocer" x="5" y="5"/>
   </s:controlBarContent>
   <s:Label text="(c) 2009, FlexGrocer" right="10" bottom="10"/>
   <s:HGroup x="0" y="0" width="100%" height="100%" id="bodyGroup">
      <s:VGroup width="100%" height="150" id="products"
```

```
            width.cartView="0" height.cartView="0"
            visible.cartView="false">
         <s:Label text="Milk" id="prodName"/>
         <s:Label text="$1.99" id="price"/>
         <s:Button label="AddToCart" id="add"/>
      </s:VGroup>
      <s:VGroup height="100%" id="cartGroup" width.cartView="100%">
         <s:Label text="Your Cart Total: $"/>
         <s:Button label="View Cart" click="this.currentState='cartView'"
            includeIn="State1"/>
         <mx:DataGrid includeIn="cartView" id="dgCart" width="100%">
            <mx:columns>
               <mx:DataGridColumn headerText="Column 1"
                  dataField="col1"/>
               <mx:DataGridColumn headerText="Column 2"
                  dataField="col2"/>
               <mx:DataGridColumn headerText="Column 3"
                  dataField="col3"/>
            </mx:columns>
         </mx:DataGrid>
         <s:Button includeIn="cartView" label="Continue Shopping"
            click="this.currentState=''"/>
      </s:VGroup>
   </s:HGroup>
</s:Application>
```

8. Zapisz plik i uruchom aplikację.

   Aplikacja powinna wyglądać i działać dokładnie tak samo jak przed refaktoryzacją. Zmianie uległ kod, dzięki czemu jest bardziej czytelny i prostszy w utrzymaniu.

Czego się nauczyłeś?

Podczas tej lekcji:

- stosowałeś kontenery i obiekty layoutu (s. 65 – 69);
- tworzyłeś aplikację w trybie edycji kodu źródłowego (s. 69 – 72);
- pracowałeś nad layoutem aplikacji w trybie projektowania (s. 72– 76);
- pracowałeś z layoutami opartymi na więzach (s. 76 – 80);
- pracowałeś ze stanami widoków (s. 81 – 84);
- sterowałeś stanami widoków (s. 84 – 86);
- refaktoryzowałeś kod aplikacji (s. 86 – 90).

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

V

W

X

Z

Ź

ADOBE® FLEX® 4
OFICJALNY PODRĘCZNIK

Adobe Flex 4 to najnowszy zestaw technologii służących do tworzenia bogatych aplikacji internetowych. Ta wydajna i bezpłatna struktura programistyczna to istne novum na rynku programów webowych. Flex pozwala na tworzenie prawdziwie dynamicznych i interaktywnych aplikacji o olbrzymich możliwościach. A to wszystko bez konieczności rezygnowania ze wspaniałej grafiki!

Dzięki książce „Adobe Flex 4. Oficjalny podręcznik. Tom I" stopniowo, krok po kroku, nauczysz się tworzyć aplikacje w językach MXML oraz ActionScript 3.0. Znajdziesz tu omówienie nowości wprowadzonych we Fleksie 4, takich jak komponenty Spark, nowe, zaawansowane techniki tworzenia skórek oraz wiele innych. Poznasz możliwości pracy w środowisku Flex Builder oraz dowiesz się, jak tworzyć interfejsy użytkownika. Zapoznasz się z wiedzą na temat kontrolek, obsługi zdarzeń, stosowania techniki „przeciągnij i upuść", tworzenia nawigacji czy też zmiany wyglądu Twojej aplikacji za pomocą stylów. Pamiętaj, że masz przed sobą najlepszy podręcznik technologii Flex, napisany przez zespół specjalistów posiadających olbrzymie doświadczenie w prowadzeniu kursów programowania, konsultacji i wszelkich działań wspomagających programistów i projektantów.

Skorzystaj z niesamowitych możliwości Fleksa 4 i stwórz swoją wyjątkową aplikację internetową!

Dołączona do książki płyta zawiera wszystkie pliki, które są potrzebne podczas kolejnych lekcji.

- Flash Builder
- Tworzenie interfejsu użytkownika
- Stosowanie kontrolek
- Obsługa zdarzeń
- ActionScript i MXML
- Tworzenie komponentów
- Siatki danych
- Rendering elementów
- Stosowanie mechanizmu „przeciągnij i upuść"
- Style i skórki

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

Nr katalogowy: 5853

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

helion.pl
księgarnia internetowa

Cena 77,00 zł

ISBN 978-83-246-2884-1

9788324628841

Informatyka w najlepszym wydaniu