

**Naucz się tworzyć efektywne aplikacje internetowe
dzięki możliwościom Adobe Flex 4!**

- Tworzenie własnych komponentów, nadawanie im stylu i skórek
- Praca nad aplikacjami z wykorzystaniem środowiska Flash Builder
- Wydajny, zoptymalizowany obiektowy język ActionScript 3.0

Zawiera CD



ADOBE® **FLEX® 4**

OFICJALNY PODRĘCZNIK



Michael Labriola, Jeff Tapper, Matthew Boles

Wstęp: Matt Chotin, menedżer produktu



» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

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!**

Spis treści

O autorach	11
Wstęp	13
Wprowadzenie	15
Lekcja 1. Wprowadzenie do bogatych aplikacji internetowych	27
Rozwój aplikacji komputerowych	27
Odejście od architektury opartej na stronie internetowej	29
Zalety bogatych aplikacji internetowych	31
Menedżerowie przedsiębiorstw	31
Przedsiębiorstwa branży IT	31
Użytkownicy końcowi	31
Technologie bogatych aplikacji internetowych	32
Asynchroniczny JavaScript i XML (AJAX)	32
Wirtualna maszyna Javy	33
Microsoft Silverlight	33
Platforma Adobe Flash	34
Czego się nauczyłeś?	37
Lekcja 2. Zaczynamy	39
Wstęp do tworzenia fleksowych aplikacji	40
Tworzenie projektu i aplikacji MXML	40
Obszar roboczy Flash Buildera	45
Uruchamianie aplikacji	48
Korzystanie z debugera programu Flash Builder	53
Przygotuj się do kolejnych lekcji	60
Czego się nauczyłeś?	62
Lekcja 3. Projektowanie interfejsu użytkownika	65
Layout aplikacji	65
Kontenery	66
Obiekty layoutu	66

Łączenie kontenerów i obiektów layoutu	67
Przewijanie zawartości	68
Budowa znaczników MXML	68
Tworzenie layoutu sklepu internetowego	69
Tworzenie zarysu layoutu w trybie edycji źródła	70
Praca nad layoutem w trybie projektowania	72
Definiowanie sekcji produktów	75
Praca z layoutami opartymi na więzach	76
Praca ze stanami widoku	81
Tworzenie stanów widoku	81
Sterowanie stanami widoku	84
Refaktoryzacja	86
Stosowanie złożonych kontenerów	88
Refaktoryzacja kodu aplikacji	89
Czego się nauczyłś?	90
Lekcja 4. Stosowanie prostych kontrolek	93
Wprowadzenie do prostych kontrolek	94
Wyświetlanie obrazów	95
Tworzenie widoku szczegółów	98
Korzystanie z wiązania danych do powiązania struktury danych z prostą kontrolką	101
Wykorzystanie kontenera Form do umieszczania prostych kontrolek	102
Czego się nauczyłś?	105
Lekcja 5. Obsługa zdarzeń	107
Wprowadzenie do obsługi zdarzeń	107
Prosty przykład	109
Obsługa zdarzenia przez funkcję języka ActionScript	110
Przekazywanie danych podczas wywoływania funkcji obsługi zdarzenia	111
Używanie danych z obiektu zdarzenia	111
Dokonywanie inspekcji obiektu zdarzenia	114
Obsługa zdarzeń systemowych	117
Zdarzenie creationComplete	117
Modyfikowanie danych w zdarzeniu creationComplete	118
Czego się nauczyłś?	120
Lekcja 6. Korzystanie ze zdalnych danych XML	123
Osadzanie danych XML	123
Umieszczenie modelu poza aplikacją	124
Wybór między obiektami a XML-em	126
Ładowanie danych XML w trakcie działania programu	129
Tworzenie obiektu HTTPService	129
Wywoływanie metody send()	130
Uzyskiwanie dostępu do otrzymanych danych	130
Problemy związane z bezpieczeństwem	131
Pobieranie danych XML za pośrednictwem obiektu HTTPService	133

Przeszukiwanie XML-a za pomocą E4X	135
Operatory E4X	136
Korzystanie z dynamicznych danych XML	141
Używanie obiektu XMLListCollection z kontrolką List	145
Czego się nauczyłeś?	147
Lekcja 7. Tworzenie klas	149
Tworzenie własnej klasy w języku ActionScript	149
Tworzenie obiektu wartości	150
Przygotowanie metody tworzącej obiekt	156
Tworzenie klas koszyka na zakupy	159
Manipulowanie danymi w klasie ShoppingCart	164
Dodawanie produktów do koszyka	164
Dodawanie pozycji czy uaktualnianie liczby sztuk?	166
Warunkowe dodawanie pozycji do koszyka	166
Czego się nauczyłeś?	171
Lekcja 8. Stosowanie wiązania danych i kolekcji	173
Zasada działania mechanizmu wiązania danych	173
Prosty przykład	174
Bardziej skomplikowany przykład	177
Mechanizm wiązania danych z perspektywy kompilatora	179
Metody dostępne set i get	180
Rozgłaszanie i nasłuchiwanie zdarzeń	181
Zdarzenia w mechanizmie wiązania danych	182
Podsumowanie informacji o wiązaniu danych	184
Korzystanie z klasy ArrayCollection	184
Wypełnianie obiektu ArrayCollection danymi	185
Korzystanie z danych zapisanych w kolekcji ArrayCollection	191
Sortowanie elementów kolekcji ArrayCollection	193
Refaktoryzacja kodu — wyszukiwanie za pomocą kursora	197
Usuwanie elementów z wykorzystaniem kursora	200
Filtrowanie kolekcji ArrayCollection	201
Refaktoryzacja klasy ShoppingCartItem	202
Czego się nauczyłeś?	204
Lekcja 9. Tworzenie komponentów	207
Wprowadzenie w tematykę komponentów MXML	208
Zasady tworzenia komponentów	209
Tworzenie własnego komponentu	209
Własne komponenty a architektura aplikacji	211
Tworzenie komponentu ShoppingView	213
Tworzenie komponentu ProductItem	218
Tworzenie komponentów zarządzających pobieraniem danych	226
Czego się nauczyłeś?	233

Lekcja 10. Korzystanie z komponentów DataGroup oraz List	235
Korzystanie z komponentów List	236
Właściwość labelFunction klasy List	236
Korzystanie z komponentów DataGroup	238
Implementacja renderera elementów	239
Zastosowanie komponentu DataGroup w klasie ShoppingView	243
Wirtualizacja	246
Implementowanie wirtualizacji	246
Wirtualizacja w komponentach List	247
Wyświetlanie produktów z wybranej kategorii	249
Filtrowanie produktów pod względem kategorii	249
Wykorzystanie zdarzenia change komponentu List	250
Czego się nauczyłeś?	251
Lekcja 11. Tworzenie i rozgłaszanie zdarzeń	253
Korzyści ze stosowania luźno powiązanej architektury	253
Rozsyłanie zdarzeń	255
Deklarowanie zdarzeń komponentu	258
Kiedy przydają się klasy niestandardowych zdarzeń	260
Tworzenie i używanie zdarzenia UserAcknowledgeEvent	261
Strumień zdarzenia i faza propagacji	264
Tworzenie i używanie klasy ProductEvent	269
Tworzenie komponentu ProductList	271
Korzystanie z komponentu ProductList	273
Korzystanie ze zdarzenia ProductEvent podczas dodawania i usuwania produktów	274
Obsługa zdarzenia COLLECTION_CHANGE	275
Czego się nauczyłeś?	277
Lekcja 12. Stosowanie siatek danych i rendererów elementów	279
Dwa zestawy kontrolki: Spark i MX	280
Komponent DataGrid i renderery elementów	280
Wyświetlanie koszyka na zakupy w kontrolce DataGrid	281
Korzystanie z komponentu CartGrid	283
Dodawanie kontrolki edycji do DataGridColumn	283
Tworzenie renderera elementów służącego do wyświetlenia informacji o produkcie	285
Tworzenie renderera elementów służącego do wyświetlenia przycisku Remove	287
Ponowne wykorzystanie klasy ProductEvent	289
Wykorzystanie właściwości labelFunction do wyświetlania wartości produktów	290
Używanie komponentu AdvancedDataGrid	292
Sortowanie zawartości kontrolki AdvancedDataGrid	292
Sortowanie w trybie zaawansowanym	293

Nadawanie stylów komponentowi AdvancedDataGrid	294
Grupowanie danych	298
Wyświetlanie danych podsumowujących	303
Czego się nauczyłeś?	311
Lekcja 13. Obsługa techniki „przeciągnij i upuść”	313
Zasada działania menedżera przeciągania i upuszczania	314
Przeciąganie i upuszczanie między dwoma komponentami DataGrid	315
Przeciąganie i upuszczanie między siatką danych i listą	317
Stosowanie mechanizmu „przeciągnij i upuść” w komponentach, które go nie obsługują	322
Przeciąganie produktu do koszyka na zakupy	327
Czego się nauczyłeś?	332
Lekcja 14. Tworzenie nawigacji	335
Wprowadzenie do nawigacji	335
Przygotowanie procesu dokonywania płatności wyświetlanego w kontenerze ViewStack	337
Integrowanie aplikacji z komponentem CheckOutView	342
Czego się nauczyłeś?	344
Lekcja 15. Formatowanie i walidacja danych	347
Podstawowe informacje o klasach formatujących i walidujących	347
Klasy formatujące	348
Klasy walidujące	348
Korzystanie z klas formatujących	349
Korzystanie z dwustronnego wiązania danych	352
Korzystanie z klas walidatorów	353
Czego się nauczyłeś?	356
Lekcja 16. Zmiana wyglądu aplikacji za pomocą stylów	359
Wprowadzenie projektu graficznego aplikacji za pomocą stylów i skórek	359
Przygotowanie aplikacji na zmiany	360
Stosowanie stylów	361
Nadawanie stylów przez atrybuty znaczników	362
Dziedziczenie stylów	364
Nadawanie stylów za pomocą znacznika <mx:Style>	365
Nadawanie stylów za pomocą plików CSS	368
Definiowanie stylów dla pozostałych elementów aplikacji	371
Zmiana stylów w trakcie działania aplikacji	377
Korzyści płynące z wczytywania stylów	378
Tworzenie pliku SWF z arkusza CSS	378
Wczytywanie arkusza stylów za pomocą klasy StyleManager	379
Przesłanie stylów we wczytanych plikach CSS	379
Czego się nauczyłeś?	379

Lekcja 17. Zmiana wyglądu aplikacji za pomocą skórek	381
Skórki komponentów Spark	381
Powiązanie między skórkami i stanami	385
Rysowanie we Fleksie	385
Definiowanie wyglądu przycisku w różnych stanach	389
Tworzenie skórki dla aplikacji	393
Czego się nauczyłeś?	397
 Lekcja 18. Tworzenie komponentów w języku ActionScript	 399
Komponenty tworzone w języku ActionScript	400
Tworzenie komponentów może być skomplikowane	400
Komponenty od środka	401
Po co tworzyć komponenty?	402
Definiowanie komponentu	403
Definiowanie interfejsu	404
Wybór klasy bazowej	405
Tworzenie klasy	406
Używanie zdefiniowanej klasy	408
Tworzenie części wizualnej	410
Określenie wymagań dotyczących skórki	410
Tworzenie skórki	412
Dodawanie funkcjonalności do komponentu	416
Asynchroniczna praca komponentów	416
Komunikowanie się za pomocą zdarzeń	421
Sprzątanie po sobie	424
Tworzenie renderera dla skórki	426
Czego się nauczyłeś?	428
 Dodatek A Przygotowanie do pracy	 432
Instalowanie oprogramowania	432
Instalowanie środowiska Flash Builder	432
Kopiowanie plików lekcji	432
Instalowanie odtwarzacza Flash Player z możliwością debugowania	433
Importowanie projektów	434
Importowanie plików lekcji	434
 Skorowidz	 437

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 wyposażać 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 SkinnableContainer 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 wymagającym 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 Flexsem 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.

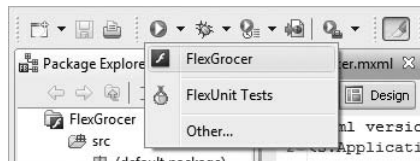
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  minWidth="955" minHeight="600">
  <fx:Declarations>
    <!-- Place non-visual elements (e.g., services, value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

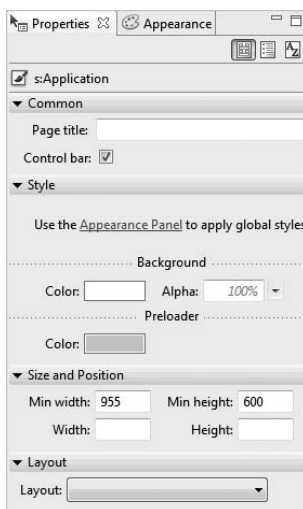
Do przełączania między trybem projektowania i trybem edycji źródła używaj przycisków umieszczonych na pasku tytułowym w górnej części okna. Po włączeniu trybu projektowania zobaczysz podgląd opracowanego okna aplikacji.

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.



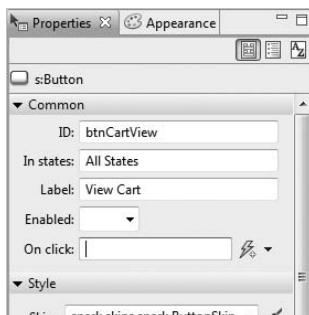
3. Kliknij przycisk *Flex Grocer*, który dodałeś w poprzednim ćwiczeniu.

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 kontrolki, 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 kontrolki. 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 jakiegos znacznika brakuje lub jest błędnie zapisany, popraw to, korzystając z poniższego kodu. Oczywiście mogą wystąpić drobne różnice w wartościach współrzędnych *x* i *y* kontroltek, 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 *VerticalLayout*, nazwa produktu jest wyświetlana nad ceną.



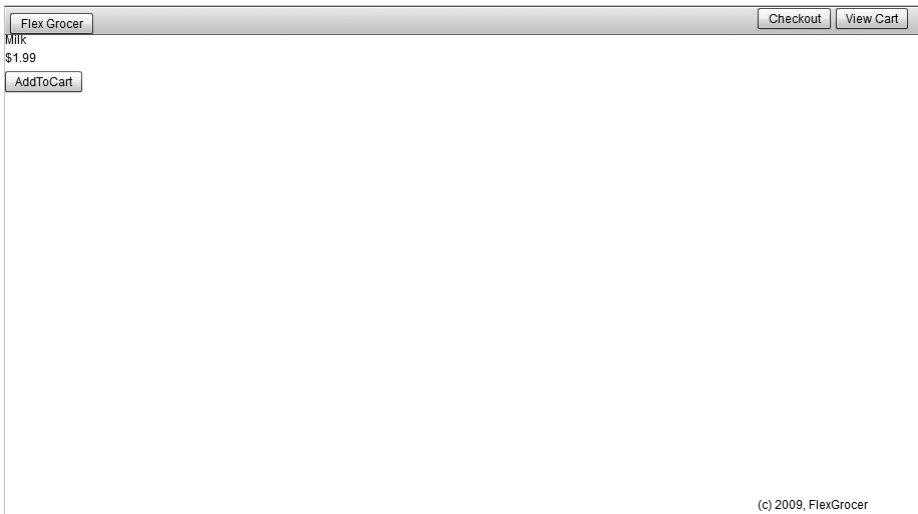
*Jeżeli otworzysz widok Outline, klikając zakładkę Outline (znajdującą się obok zakładki Components, z której już korzystałeś), będziesz mógł zobaczyć hierarchię elementów tworzących aplikację. Głównym elementem jest znacznik <s:Application>, w którym są zawarte elementy potomne: etykieta (*Label*) i kontener *bodyGroup*, a także właściwości *controlBarContent* i *controlBarLayout*. Widoczne są również inne składniki elementów *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.

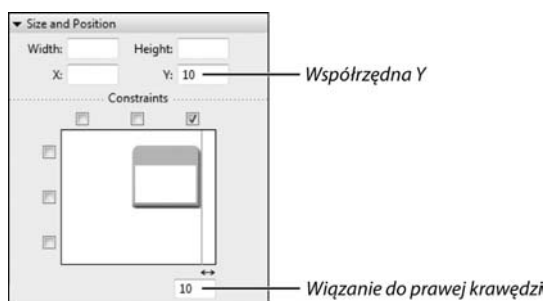


Wszystkie więzy ustawiane są względem krawędzi kontenera, jeżeli zastosowano w nim pozycjonowanie bezwzględne (obiekt *BasicLayout*), więc nie mogą być ustawiane względem innych kontroltek lub kontenerów.

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.

- 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 odszukiwaniem 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>
```

- 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.

- 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 kontroltek 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.

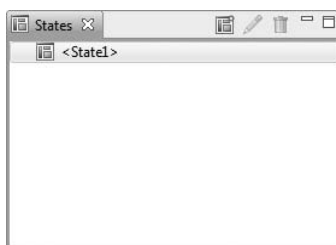
Dodatkowe stany w MXML-u to zmodyfikowane ustawienia widoku podstawowego lub innych stanów.

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>**.

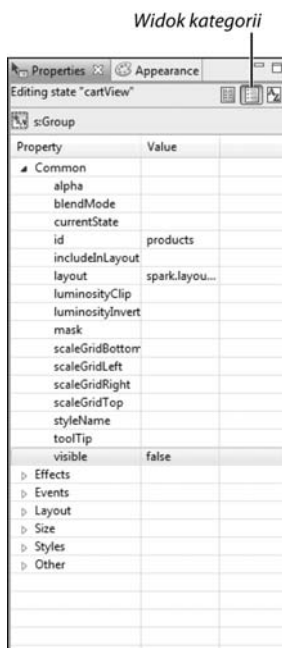
Nowy stan tworzy się przez kliknięcie przycisku *New State* u góry panelu *States* lub przez kliknięcie prawym przyciskiem myszy w panelu i wybranie z podręcznego menu opcji *New*. W widoku *cartView* będą wyświetlane szczegółowe informacje o wszystkich produktach umieszczonych przez użytkownika w koszyku.

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 warto 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 *OnClick* 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>

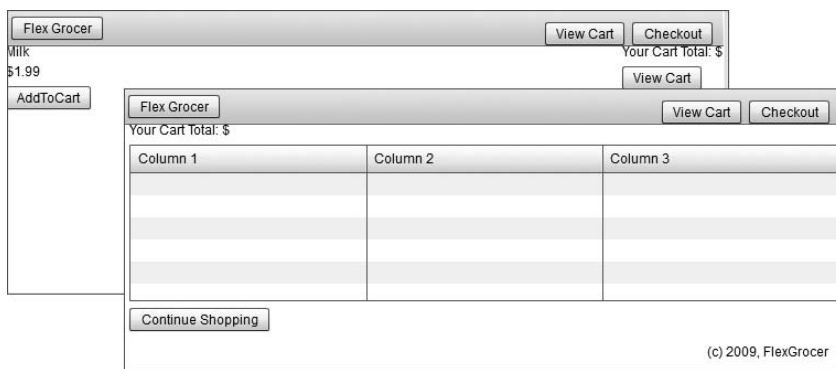
    <s:controlBarContent>
        <s:Button id="btnCheckout" label="Checkout" right="10" y="10"/>
        <s:Button id="btnCartView" label="View Cart" right="90" y="10"
            click:State1="this.currentState='cartView'"/>
        <s:Button label="Flex Grocer" x="5" y="5"/>
    </s:controlBarContent>
    <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:Group>
    </s:Group>
</s:Application>
```

```

        <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: poznasz 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 <code>layout</code>	Wersja ze złożonymi kontenerami
<pre><s:Group> <s:layout> <s:HorizontalLayout/> </s:layout> <s:Button label="1"/> <s:Button label="2"/> <s:Button label="3"/> </s:Group></pre>	<pre><s:HGroup> <s:Button label="1"/> <s:Button label="2"/> <s:Button label="3"/> </s:HGroup></pre>
<pre><s:Group> <s:layout> <s:VerticalLayout/> </s:layout> <s:Button label="1"/> <s:Button label="2"/> <s:Button label="3"/> </s:Group></pre>	<pre><s:VGroup> <s:Button label="1"/> <s:Button label="2"/> <s:Button label="3"/> </s:VGroup></pre>

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 `VerticalLayout` 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

..., 140, 190
.NET, 32
@, 140, 353
@Embed, 97, 384
@font-face, 372
@namespace, 372
[Bindable], 152, 175, 176, 177, 179, 181, 194
[Event], 258
[inherited], 115
{}, 175, 180
++, 167
<![CDATA[]>, 110
<cross-domain-policy>, 132
<fx:Component>, 287, 288
<fx:Declarations>, 44, 101, 133, 134, 214, 224, 232, 282
<fx:Metadata>, 259, 268, 273, 289
<fx:Model>, 101, 124, 126
<fx:Script>, 110, 113, 134, 213, 216, 219, 256, 289, 290
<fx:Style>, 367
<fx:XML>, 128, 133, 155, 157, 189
<mx:AdvancedDataGrid>, 308
<mx:AdvancedDataGridColumn>, 295, 298
<mx:AdvancedDataGridRendererProvider>, 307
<mx:columns>, 281, 282
<mx:Component>, 287
<mx:CurrencyFormatter>, 350
<mx:DataGrid>, 281, 315
<mx:DataGridColumn>, 281, 287
<mx:dataProvider>, 299
<mx:fields>, 305
<mx:Form>, 102
<mx:FormHeading>, 102
<mx:FormItem>, 102, 104, 353
<mx:Grouping>, 299
<mx:GroupingField>, 299, 300
<mx:HorizontalList>, 315
<mx:Image>, 97, 222, 285, 413
<mx:itemRenderer>, 287, 288
<mx:Label>, 95
<mx:List>, 315
<mx:Menu>, 315
<mx:PrintDataGrid>, 315
<mx:rendererProviders>, 307
<mx:Script>, 163
<mx:Style>, 365
<mx:summaries>, 304
<mx:SummaryField2>, 305
<mx:SummaryRow>, 304
<mx:TileList>, 315
<mx:Tree>, 315
<mx:ViewStack>, 336
<mx:ZipCodeValidator>, 354
<s:Application>, 43, 44, 51, 210, 217
<s:BasicLayout>, 70
<s:Button>, 59, 71
<s:controlBarContent>, 71
<s:controlBarLayout>, 70
<s:DataGroup>, 244, 414
<s:dataProvider>, 210
<s:GradientEntry>, 390
<s:Group>, 67, 69, 209, 213
<s:HTTPService>, 133, 142
<s:Label>, 49, 50, 222
<s:layout>, 69, 245
<s:Line>, 415
<s:List>, 209, 210, 315
<s:MXItemRenderer>, 285, 288
<s:Rect>, 387
<s:RichText>, 100, 157
<s:Scroller>, 68, 414
<s:SparkSkin>, 384
<s:State>, 215
<s:states>, 98
<s:VerticalLayout>, 69, 245, 288
<s:VGroup>, 98

A

AATC, 24
ACA, 24
acceptDragDrop(), 323, 325, 329

- ACE, 24
- ACI, 24
- ActionScript, 15, 32, 110, 149
- ActionScript 3.0, 36
- ActionScript Class, 159
- ActionScript Virtual Machine, 36
- addData(), 322, 324
- addEventListener(), 182, 183, 228, 231, 265, 422, 424, 425
- addItem(), 321, 326
- addProductHandler(), 274
- Adobe, 15
- Adobe Community Help, 22
- Adobe Flash, 34
- Adobe Flash Builder, 40
- Adobe Flash Player, 35
- Adobe Flex, 93
- Adobe Integrated Runtime, 35
- adres URL, 129
- AdvancedDataGrid, 280, 292, 295, 303, 308
 - AdvancedDataGridColumn, 297
 - creationComplete, 300, 302
 - dataField, 304
 - dataProvider, 299
 - funkcja stylu, 297
 - GroupingCollection2, 299, 300
 - grupowanie danych, 298
 - grupowanie danych za pomocą kodu
 - ActionScript, 300
 - grupowanie danych za pomocą znaczników, 299
 - kolumny, 295
 - kryterium sortowania, 294
 - nadawanie stylów, 294
 - podsumowanie, 303
 - renderer elementów, 307
 - rendererProvider, 307
 - sortExpertMode, 292, 293, 294
 - sortowanie w trybie zaawansowanym, 293
 - sortowanie zawartości, 292
 - style, 294
 - style kolumn, 295
 - style komórek, 297
 - style wierszy, 296
 - styleFunction, 296
 - summaryPlacement, 304
 - wyświetlanie danych podsumowujących, 303
 - wyświetlanie danych podsumowujących
 - za pomocą języka ActionScript, 309
 - wyświetlanie danych podsumowujących
 - za pomocą znaczników, 303
 - zmiana sposobu wyświetlania, 307
- AdvancedDataGridColumn, 295, 297
 - dataField, 298
- AdvancedDataGridRendererProvider, 307, 308, 309
- AIR, 35
- AIR Community Help, 22
- AJAX, 32
- aktualizacja widoku, 174
- aktywność komponentów, 104
- Alert, 344
- alternatywna implementacja kontenerów, 88
- API, 93
- aplety Javy, 32
- aplikacje, 27
 - AJAX, 33
 - biznesowe, 27
 - e-commerce, 69
 - Flex, 40
 - internetowe, 29
 - MXML, 40, 103
 - oparte na HTML, 28
 - RIA, 31
- Application, 66, 70, 118, 183, 256, 266, 267
 - dispatchEvent(), 183
 - skórki, 393
- architektura aplikacji, 211
- architektura aplikacji biznesowych, 27
- architektura MVC, 212, 213
- architektura oparta na stronie internetowej, 28, 29
- architektura zorientowana na usługę, 29
- argumenty, 153
- Array, 164, 184, 185, 190, 194, 195, 201
- ArrayCollection, 184, 185, 189, 190, 193, 197, 201, 214, 239, 299, 300, 309, 316
 - filtrowanie kolekcji, 201
 - getItemAt(), 191
 - odczytywanie danych, 191
 - sortowanie elementów, 193
 - wypełnianie danymi, 185
- ArrayList, 185, 239
- AS3 Drawing API, 385
- ASDoc, 59
- Asynchroniczny JavaScript i XML, 32
- asynchroniczny model komponentów, 416
- AVG, 306
- AVM, 36
- AVM2, 36

B

- background-color, 365
- BasicLayout, 67, 76
- beginFill(), 386, 387
- bezpieczeństwo, 131

bezzastanowy charakter sieci, 30
biblioteki AJAX, 33
BitmapFill, 389
BitmapImage, 331
blok CDATA, 113
bodyGroup, 74
bogate aplikacje internetowe, 15, 27
 zalety, 31
Boolean, 109, 152
BorderContainer, 66
breakpoint, 53
Breakpoints, 55, 58
budowanie aplikacji, 48
Build Automatically, 48
Button, 71, 118, 265, 266, 280
ButtonSkin, 382, 388

C

CamelCase, 365
CartGrid, 283
Cascading Style Sheet, 361
CDATA, 113
cel upuszczenia, 314
certyfikaty firmy Adobe, 24
change, 250
CheckoutView, 342
ciasne powiązania, 254
class, 151, 152
clear, 109
click, 108, 109, 267
clipAndEnableScrolling, 285
clone(), 261, 262
COLLECTION_CHANGE, 275
CollectionEvent, 276
CollectionEvent.COLLECTION_CHANGE, 276, 419
color, 361, 366
ComboBox, 123
commitProperties(), 418, 420, 421
Community Help, 22
Compile CSS to SWF, 378
completeOrder, 341
component-based development, 207
Components, 74, 80
Constraints, 78
controlBarLayout, 70, 71
cookies, 30
cornerRadius, 365
COUNT, 306
Create Watch Expression, 127, 144, 176

createCursor(), 198, 200
creationComplete, 112, 117, 118, 300, 302
 modyfikacja danych, 118
CreditCardValidator, 349
cross-domain policy file, 132
crossdomain.xml, 132
CSS, 359, 361
CSS Inheritance, 364
CurrencyFormatter, 348, 349
currentState, 84, 100
curveTo(), 386
CustomerInfo, 339
czcionki, 371

D

dane, 173, 212
dane XML, 123, 157
 bezpieczeństwo, 131
 dane XML, 145
 dynamiczne dane, 141
 elementy, 137
 główny węzeł, 137
 HTTPService, 129
 ładowanie danych w trakcie działania programu, 129
 osadzanie, 123
 pobieranie, 133
 przeszukiwanie, 135
 węzły, 137
 XMLList, 137
Data Transfer Object, 150
dataField, 281, 286, 305
dataForFormat(), 320, 322, 324, 326, 330
DataGrid, 83, 84, 279, 280, 281, 284
 dataField, 281
 DataGridColumn, 281
 dragEnabled, 316
 dropEnabled, 317
 editable, 281, 282
 mapowanie kolumny zbioru danych do zadanej kolumny, 281
 przeciagnij i upuść, 315
 renderer elementów, 285
 składnia, 281
 tworzenie niestandardowego renderera elementu, 285
 variableRowHeight, 286
 wyświetlanie koszyka na zakupy, 281
DataGridColumn, 281, 282
 dataField, 282, 286
 dodawanie kontrolki edycji, 283
 editable, 282

- DataGridColumn
 - headerText, 282
 - itemRenderer, 286
 - tworzenie renderera, 287
- DataGroup, 66, 235, 238, 239, 243, 272
 - dataProvider, 239
 - DefaultItemRenderer, 239
 - IDataRenderer, 240
 - implementacja renderera elementów, 239
 - renderer elementów, 239
 - stosowanie, 243
 - wirtualizacja, 246
 - zbiór danych, 239
- dataProvider, 146, 235, 236, 239, 299, 301
- DataRenderer, 244, 248, 426
- Date, 123, 262
- DateField, 104
- DateFormatter, 348
- DateValidator, 349
- Debug, 55, 114, 115, 127
 - Breakpoint Properties, 59
 - Breakpoints, 55, 58
 - Expressions, 55
 - pułapki, 55, 58
 - stan zmiennych, 55
 - Step Into, 55, 58
 - Step Over, 56, 59
 - Step Return, 59
 - Variables, 55, 56, 57
 - właściwości pułapki, 59
 - wyrażenia, 55
- debuger, 53
- debugowanie, 46, 142
- default state, 81
- DefaultItemRenderer, 239
- definicja metody, 151
- definicja pakietu, 152
- definiowanie
 - grafika, 386
 - komponenty, 403
 - przestrzenie nazw, 44
 - sekcja produktów, 75
 - style, 371
 - wygląd przycisku w różnych stanach, 389
- deklaracja
 - klasy, 152
 - przestrzenie nazw, 44, 372
 - właściwości klasy, 151
 - zdarzenia komponentu, 258
- deklaracja typu dokumentu, 125
- deklaracja typu dokumentu XML, 43
 - description, 118, 192
- Design, 124
- DHTML, 28, 29, 32
- dispatchEvent(), 182, 183, 255, 257, 260, 289
- display list, 264
- Document Object Model, 29
- dodawanie pozycji do koszyka, 162
- doDrag(), 323, 328, 332
- doDragEnter(), 325
- dokonywanie płatności, 337
- DOM, 29, 32
- domyślny konstruktor, 153
- dostarczanie danych do komponentu, 210
- dostęp do danych XML, 129, 139
- Drag initiator, 314
- Drag proxy, 314
- Drag source, 314
- dragComplete, 318
- dragDrop, 319
- dragEnabled, 315, 316, 317
- dragEnter, 319
- dragEvent, 318
- DragEvent, 112
- DragEvent.DRAG_DROP, 423
- DragEvent.DRAG_ENTER, 422, 423
- dragExit, 319
- DragManager, 323, 328
 - acceptDragDrop(), 323, 325
 - doDrag(), 323, 328, 332
- dragOver, 319
- dragSource, 325
- DragSource, 314, 316, 320, 322, 324
 - addData(), 322
 - dataFormat(), 322
 - hasFormat(), 322
- drawRect(), 387
- Drop target, 314
- DropDownList, 362, 363, 365, 367, 368
- dropEnabled, 315, 317
- DTO, 150
- dwustronne wiązanie danych, 352
- Dynamic HTML, 28
- dynamiczne dane XML, 141
- dziedziczenie stylów, 364

E

- E4X, 123, 128, 135, 186
 - @, 140
 - dostęp do danych XML, 139
 - filtr predykatowy operujący na atrybutach, 140

- filtrowanie predykatowe, 139
- operator atrybutu, 140
- operator potomków, 140
- operatory, 136
- poszukiwanie potomków, 190
- ECMAScript for XML, 123
- editorDataField, 284
- edytor, 45
- eksplorator pakietów, 124, 151
- elementy wizualne aplikacji Fleksa, 94
- elementy XML, 137
- EmailValidator, 349
- Embed, 384
- embedAsCFF, 372
- endFill(), 386
- etykiety, 103
- event, 182
- Event, 112, 182, 253, 257
- event.timestamp, 264
- EventDispatcher, 255, 265
- events, 267
- Expressions, 55, 127
- Extending Flex Components, 197
- eXtensible Application Markup Language, 34

F

- FAQ, 23
- faza propagacji zdarzenia, 264
- FedEx Custom Critical, 32
- fields, 195
- fill, 389
- filterFunction, 202
- filtrowanie kolekcji, 201
- filtrowanie predykatowe, 139
- filtrowanie produktów pod względem kategorii, 249
- findAny(), 198, 199
- findFirst(), 198, 199, 200
- findLast(), 199
- Flash, 34
- Flash Builder, 18, 39, 432
 - Build Automatically, 48
 - Debug, 55
 - debuger, 53
 - debugowanie, 46
 - Flash, 46
 - Flash Builder Project, 53, 61
 - Flash Debug, 46
 - Import Flash Builder, 61
 - importowanie projektów, 434
 - Local History, 51

- numery wierszy kodu, 47
- obszar roboczy, 45
- Open Perspective, 46
- Package Explorer, 46
- perspektywy, 46
- Project, 48
- projekt, 40
- projektowanie interfejsu użytkownika, 65
- Replace With/Local History, 51
- Run, 48
- uruchamianie aplikacji, 48
- ustawienia historii lokalnej, 51
- usuwanie projektu, 60
- widoki, 46
- Flash Builder 4, 36
- Flash Catalyst, 37
- Flash Debug, 135, 142
- Flash Player, 32, 131, 416, 433
- Flash Player 10, 36
- Flash Professional, 35
- flash.events.Event, 259, 260, 261, 268
- flash.events.EventDispatcher, 255, 265
- Flex, 35
- Flex 1, 15
- Flex 2, 15
- Flex 3, 15
- Flex 4, 15
- Flex Grocer, 71, 73
- Flex MX, 44
- Flex SDK, 36, 94
- Flex Spark, 44
- flex-config.mxml, 44
- FlexEvent, 119
- FlexGrocer, 39
- FlexGrocer.mxml, 45, 70
- fontFamily, 361, 371
- fontSize, 362
- fontStyle, 362
- fontWeight, 295, 362
- for, 166, 167
- for each...in, 190
- Form, 93, 102, 280
- format(), 348, 350
- formatowanie danych, 347, 349
- Formatter, 348
- FormHeading, 103, 104
- FormItem, 104, 353
- formularze, 102
 - kontrolki, 102
- FP10, 36
- framework Flex, 93
- funkcja nasłuchująca, 108

funkcja obsługi zdarzenia, 108, 112, 256
przekazywanie argumentów, 111
funkcje, 110, 151
fx, 44
FXG, 385, 386

G

Generate Click Handler, 256
Generate Getter/Setter, 242, 407
get, 180, 241
getFocus(), 104
getImageAt(), 192
getItemAt(), 191
GIF, 95
Google Maps, 33
GradientEntry, 396
grafika wektorowa, 386
graphics, 386
Group, 66, 69, 74, 213, 219, 238, 266, 339
Grouping, 300, 301
fields, 301
GroupingCollection, 301
GroupingCollection2, 299, 300, 301, 303, 309
GroupingField, 300, 301, 304, 310
summaries, 304
grupowanie danych, 298
grupowanie za pomocą kodu ActionScript, 300

H

hasFormat(), 322, 325, 329
HGroup, 89, 118, 214
hierarchia komponentów, 208
horizontalAlign, 248
HorizontalLayout, 67, 89, 213, 214, 219, 280, 285
HostComponent, 411
HTML, 28
HTTP, 28, 30
HTTPService, 129, 131, 134, 141, 142, 185, 186, 226, 265, 316
dostęp do otrzymanych danych, 130
lastResult, 130
pobieranie danych XML, 133
result, 131
send(), 130
wysyłanie żądań, 130
HyperText Transfer Protocol, 28

I

ID, 95
IDataRenderer, 240, 241
IDE, 36, 39
identyfikatory
ID, 95
URI, 44
UUID, 60
IDropInListItemRenderer, 284
IFill, 389
IFrames, 32
ItemRenderer, 248
Image, 95
imageName, 222, 286
implementacja
kontener, 88
renderer elementów, 239
technika „przeciągnij i upuść”, 315
wirtualizacja, 246
import, 119, 154
Import Flash Builder, 61
importowanie klas, 154
importowanie projektów, 434
INavigatorContent, 336
includeIn, 100
inicjator przeciągania, 314
initDG(), 300, 301, 309
inkrementacja, 167
inspekcja obiektu zdarzenia, 114
instalacja odtwarzacza Flash Player z możliwością debugowania, 433
instalacja środowiska Flash Builder, 432
instancje klasy, 68
interfejs komponentu, 404
interfejs użytkownika, 65, 212
kontenery, 66
łączenie kontenerów i obiektów layoutu, 67
obiekty layoutu, 66
przewijanie zawartości, 68
stany widoku, 81
invalidateProperties(), 418, 420
isLowFat, 157
isOrganic, 157
itemEditor, 284
itemRenderer, 235, 272, 286
ItemRenderer, 248
items, 164
UIComponent, 325, 328

J

JavaScript, 29, 32
język ActionScript, 149
język HTML, 28
język MXML, 42, 50
JIT, 36
JPG, 95
Just In Time, 36

K

kaskadowe arkusze stylów, 361
-keep-generated-actionscript, 150
klasa bazowa, 230
 komponenty, 405
klasy, 149, 214, 406
 deklaracja, 152
 deklaracja właściwości, 151
 domyślny konstruktor, 153
 Event, 112
 importowanie, 154
 konstruktor, 151, 156
 lokalizacja, 151
 metody, 151
 MouseEvent, 114
 this, 153
 tworzenie, 149
 właściwości, 151
klasy CSS, 361
klasy formatujące, 347, 348, 349
klasy niestandardowych zdarzeń, 260
klasy potomne, 112
klasy walidatorów, 348, 353
klient-serwer, 28
kod ActionScript, 109, 110, 150
kod pocztowy, 353
kodowanie znaków, 125
kolekcje, 173, 184
 ArrayCollection, 184
 filtrowanie danych, 201
 kursory, 197
 odczytywanie danych, 191
 przetwarzanie, 190
 sortowanie elementów, 193
 usuwanie elementów, 200
 wypełnianie danymi, 185
 wyszukiwanie, 197
 XMLListCollection, 185
kompilacja JIT, 36
komponent rozgłaszający zdarzenie, 112

komponenty, 173, 207
 ActionScript, 400
 AdvancedDataGrid, 292
 Application, 393
 architektura aplikacji, 211
 architektura MVC, 212
 asynchroniczny model komponentów, 416
 CartGrid, 283
 CheckoutView, 342
 commitProperties(), 418
 część wizualna, 410
 DataGrid, 279, 280
 DataGroup, 235, 238
 definicja dokumentu XML, 209
 definiowanie interfejsu, 404
 definiowanie komponentu, 403
 deklaracja zdarzeń, 258
 dodawanie funkcjonalności, 416
 dostarczanie danych, 210
 FormHeading, 103, 104
 hierarchia komponentów, 208
 HTTPService, 129
 interfejs, 404
 invalidateProperties(), 418
 klasa bazowa, 405
 komunikacja, 421
 List, 210, 235, 236
 metadane, 259
 MX, 44
 MXML, 208
 nasłuchiwanie zdarzeń, 424
 NavigatorContent, 336
 nazwy, 209
 Panel, 266
 ProductItem, 218
 ProductList, 271
 przewijanie zawartości, 414
 rozsyłanie zdarzeń, 255
 Scroller, 414
 ShoppingList, 399
 ShoppingView, 213, 218, 402
 SkinnableComponent, 401
 skórki, 381, 401, 410
 Spark, 44
 sposób definiowania komponentu, 401
 stosowanie, 211
 tworzenie, 209, 271, 399, 400
 tworzenie części wizualnej, 410
 tworzenie klasy, 406
 tworzenie skórki, 412
 UIComponent, 208, 401

- komponenty
 - VGroup, 98
 - ViewStack, 336
 - WarningDialog, 268
 - wielokrotne wykorzystanie, 211
 - zasady tworzenia, 209
 - zdarzenia, 275, 421
 - zestawy komponentów, 401
 - komponenty nawigacyjne, 336
 - komponenty zarządzające pobieraniem danych, 226
 - komunikaty, 344
 - konstruktor, 151, 156
 - domyślny konstruktor, 153
 - kontenery, 66
 - Application, 66
 - BorderContainer, 66
 - Form, 102
 - Group, 66, 238
 - kontenery nawigacyjne, 336
 - NavigationContent, 66
 - Panel, 66
 - SkinnableContainer, 66
 - kontroler, 212
 - kontrolki, 93, 94
 - AdvancedDataGridColumn, 295
 - DropDownList, 362
 - formularze, 102
 - HorizontalLayout, 280
 - ID, 95
 - Image, 95
 - Label, 94
 - MX, 280
 - powiązanie struktury danych, 101
 - przeciągnij i upuść, 315
 - RichText, 94, 99
 - Spark, 280
 - TextArea, 94
 - TextInput, 94, 352
 - VerticalLayout, 280
 - wiązanie danych, 95, 101
 - zestawy kontrolek, 280
 - konwencja CamelCase, 365
 - konwersja danych XML na obiekt, 126
 - koszyk na zakupy, 159, 403
 - aktualizacja zawartości, 275
 - CartGrid, 283
 - dodawanie pozycji, 162, 164, 166, 271, 274
 - lista produktów, 236
 - manipulowanie danymi, 164
 - pozycje, 161
 - ProductEvent, 269, 274
 - ProductList, 271
 - przeciąganie produktu, 327
 - ShoppingCart, 162, 163
 - ShoppingCartItem, 159, 202
 - ShoppingList, 399, 404
 - ShoppingView, 402
 - sprawdzanie występowania elementu, 167
 - szukanie pozycji zamówienia, 166
 - uaktualnianie liczby wystąpień produktów, 166, 168
 - usuwanie pozycji, 200, 271, 274
 - warunkowe dodawanie pozycji, 166
 - wyświetlanie w kontrolce DataGrid, 281
 - wyświetlanie wartości produktów, 290
 - kursory, 197
 - obsługa, 199
 - tworzenie, 198
 - usuwanie elementów, 200
- ## L
- label, 68, 71, 306
 - Label, 70, 75, 94, 118, 150, 280, 323
 - przeciągnij i upuść, 322
 - style, 361
 - labelField, 146, 236
 - labelFunction, 236, 290
 - lastResult, 130, 131
 - layout, 67, 88, 415
 - Layout, 74, 103
 - useVirtualLayout, 246
 - layout aplikacji, 65
 - layout oparty na więzach, 76
 - layout sklepu internetowego, 69
 - LayoutObject, 88
 - lekki klient, 28
 - Line, 415
 - LinearGradient, 389, 390, 396
 - lineStyle(), 387
 - lineTo(), 386
 - List, 123, 145, 150, 210, 235, 236
 - change, 250
 - dataProvider, 236
 - labelFunction, 236
 - przygotowanie wyświetlanego tekstu, 236
 - selectedIndex, 247
 - selectedItem, 247
 - wirtualizacja, 247
 - XMLListCollection, 145
 - zaznaczanie elementów, 250
 - zbiór danych, 236
 - lista wyświetlania, 264

logika biznesowa, 162
logika odpowiedzialna za obsługę zdarzeń, 212
lokalizacja klasy, 151
luźno powiązana architektura, 253

Ł

ładowanie danych XML w trakcie działania programu, 129
łańcuchy znakowe, 109
łączenie kontenerów i obiektów layoutu, 67

M

Mac OS, 431
Macromedia, 15
manipulowanie danymi w koszyku na zakupy, 164
MAX, 306
mechanizm bezpieczeństwa Flash Playera, 132
mechanizm dwustronnego wiązania danych, 352
mechanizm wiązania danych, 173
mechanizm wirtualizacji, 246
menedżer przeciągania i upuszczania, 314
metadane, 259
metody, 151
 addEventListener(), 182, 183
 argumenty, 153
 dispatchEvent(), 182, 183
 konstruktor, 156
 metody fabrykujące, 156
 metody statyczne, 156
 parametry, 153
 toString(), 154
metody dostępowe, 180, 241, 407
 get, 180
 set, 180
Microsoft Expression Studio, 34
Microsoft Silverlight, 33
MIN, 306
minHeight, 50
miniaturki obrazu, 285
minWidth, 50
model, 173, 212
model bezpieczeństwa, 132
model dokumentu HTML, 32
model programowania oparty na zdarzeniach, 107
Model-View-Controller, 212
model-widok-kontroler, 212
modyfikacja danych w zdarzeniu creationComplete, 118
mouseDown, 318, 323, 324, 328
MouseEvent, 112, 113, 114, 253, 256, 318

MouseEvent.CLICK, 423
mouseMove, 318, 324
mouseOver, 100
moveNext(), 198
movePrevious(), 198
moveTo(), 386
multipleSelection, 321
MVC, 212
 kontroler, 212
 model, 212
 przepływ informacji, 212
 widok, 212
MX, 280
mx.collections.Sort, 195
mx.collections.XMLListCollection, 143
mx.core.IDataRenderer, 241
mx.events.CollectionEvent, 276
mx.formatters.CurrencyFormatter, 348
mx.formatters.DateFormatter, 348
mx.formatters.NumberFormatter, 348
mx.formatters.PhoneFormatter, 348
mx.formatters.ZipCodeFormatter, 348
mx.rpc.events.ResultEvent, 134
mx.rpc.http.mxml.HTTPService, 230
mx.utils.ObjectProxy, 127
mx.validators.Validator, 355
MXItemRenderer, 285, 288, 289
MXML, 15, 32, 36, 42, 43, 50, 109
 instancje klasy, 68
 komponenty, 208
 stany widoku, 81
MXML Application, 103

N

nadawanie stylów, 294
 atributy znaczników, 362
 pliki CSS, 368
nasłuchiwanie zdarzeń, 108
nasłuchiwanie zdarzeń, 181, 182, 424
NavigationContent, 66
NavigatorContent, 336
nawiasy klamrowe, 175, 180
nawigacja, 335
 INavigatorContent, 336
 komponenty nawigacyjne, 336
 kontenery nawigacyjne, 336
 NavigatorContent, 336
 selectedChild, 336
 selectedIndex, 336
 ViewStack, 336, 337

nazwy funkcji obsługi zdarzeń, 114
nazwy stylów, 365
nazwy zdarzeń, 112
new, 153, 157
New ActionScript Class, 226, 230, 406
New MXML Component, 281, 285
New MXML Skin, 383, 394, 412
New State, 81
niejawne tworzenie metod dostępowych set i get, 180
niestandardowe podsumowanie danych, 306
null, 167
Number, 109, 123, 152, 160
NumberFormatter, 348
NumberValidator, 349
NumericStepper, 284

O

obiekt zdarzenia, 111, 112
obiekty, 149, 214
 konstruktor, 156
 metody dostępne, 180
 tworzenie, 153, 156
obiekty layoutu, 66
obiekty pośredniczące, 185
obiekty transferu danych, 150
obiekty wartości, 150
Object, 184, 185, 236, 241
Object-Oriented Programming, 17
ObjectProxy, 128, 185
obrazy, 95, 222, 286
obserwowanie wyrażenia, 176
obsługa JavaScript, 32
obsługa obiektów wyświetlanych na ekranie, 264
obsługa techniki „przeciągnij i upuść”, 313
obsługa zdarzeń, 107
 funkcje języka ActionScript, 110
 obsługa zdarzeń systemowych, 117
obszar roboczy Flash Buildera, 45
OkayCancelGroup, 268
OOP, 17, 149
Open Font Library, 371
operation, 305
operator ++, 167
operator potomków, 140, 190
operatory E4X, 136
OrderEvent, 344
osadzanie czcionek, 371, 372
osadzanie danych XML, 123
 umieszczenie modelu poza aplikacją, 124
Outline, 78
override, 262

P

package, 151, 152
Package Explorer, 46, 124, 151, 261
paddingLeft, 362
paddingRight, 362
pakiety, 152
Panel, 66, 266
parametry, 153
partAdded(), 421, 422
partRemoved(), 421
pasek sterowania, 70
perspektywy, 46
PhoneFormatter, 348
PhoneNumberValidator, 349
piaskownica, 132
platforma Adobe Flash, 34
plik klasy ActionScript, 159
plik manifestu, 44
pliki CSS, 368
pliki międzydomenowych zasad bezpieczeństwa, 132
pliki SWF, 49
pliki XML, 123
płatności, 337
pływające ramki, 32
PNG, 95
pobieranie danych, 226
 dane XML, 133
podglądanie wygenerowanego kodu ActionScript, 150
podpowiadanie kodu, 109
podstawowy stan, 81
podsumowanie danych, 306
pola formularza, 102
pole tekstowe, 95
poszukiwanie potomków, 190
pośredniczenie, 185
potomek, 66
powiązania danych, 109
powiązanie struktury danych z prostą kontrolką, 101
powielanie kodu, 87
preventDefault(), 321
procedura obsługi zdarzenia, 256
Product, 151, 152
ProductEvent, 269, 274, 289
ProductItem, 218
ProductList, 271, 273
ProductService, 249
produktu, 218
programowanie oparte na komponentach, 207
programowanie po stronie serwera, 108
programowanie sterowane zdarzeniami, 107

programowanie zorientowane obiektowo, 17, 149
projekt, 39, 40
projekt graficzny aplikacji, 359
projektowanie interfejsu użytkownika, 65
Properties, 72, 73
proste kontrolki, 94
protokół HTTP, 28
proxying, 185
przebieg działania aplikacji, 107
przeciąganie, 314
przeciągnij i upuść, 28, 313, 421
 acceptDragDrop(), 325
 cel upuszczenia, 314
 dataForFormat(), 320
 DataGrid, 315
 dragComplete, 318
 dragDrop, 319
 dragEnabled, 315, 316, 317
 dragEnter, 319
 dragExit, 319
 DragManager, 323, 328
 dragOver, 319
 dragSource, 325
 DragSource, 314, 316, 320, 322
 dropEnabled, 315, 317
 fazy, 314
 hasFormat(), 325
 implementacja, 327
 inicjalizacja, 314
 inicjator przeciągania, 314
 IUIComponent, 328
 komponenty bez funkcji rozszerzonych, 322
 kontrolki, 315
 Label, 322
 menedżer przeciągania i upuszczania, 314
 mouseDown, 318, 323, 324
 mouseMove, 318, 324
 obiekt zastępczy, 331
 przeciąganie, 314
 przeciąganie między siatką danych i listą, 317
 przeciąganie produktu do koszyka na zakupy, 327
 upuszczanie, 314
 zastępczy obiekt przeciągania, 314
 zdarzenia celu przeciągania, 319
 zdarzenia inicjatora przeciągania, 318
 zdarzenia przeciągania, 318
 źródło przeciągania, 314
przeglądarki internetowe, 28
przekazywanie danych podczas wywoływania funkcji
 obsługi zdarzenia, 111
przekazywanie danych w obiektach zdarzeń, 260

przełączanie aktywności komponentów, 104
przepływ informacji w architekturze MVC, 212
przesłanianie stylów we wczytanych plikach CSS, 379
przestrzeń nazw, 44, 119, 372
 fx, 44
 mx, 44
 s, 44
przeszukiwanie XML, 135
przetwarzanie
 kolekcje, 190
 tablice, 166
przewijanie zawartości, 68, 414
pseudoselektory, 376
public, 214
pułapki, 53, 55
push(), 164, 190

R

RadialGradient, 389
refaktoring, 254
refaktoryzacja, 86, 89, 197, 202, 254
 stosowanie złożonych kontenerów, 88
refresh(), 196, 202, 300, 302, 309
remove(), 200
removeEventListener(), 424, 425
removeItem(), 289
renderer elementów, 235, 239, 279, 280, 307, 426
 implementacja, 239
 tworzenie, 285
 wyświetlanie informacji o produkcie, 285
 wyświetlanie przycisku Remove, 287
rendererProvider, 307
required, 353
result, 129, 131, 265
ResultEvent, 112, 134
resultFormat, 142, 230
return, 157, 168, 189
Review, 341
RIA, 15, 31
Rich Internet Application, 15
RichText, 94, 99, 118, 192
rollover, 98
rollOverColor, 362, 363, 365, 366
root node, 137
rozgłaszacz zdarzeń, 108
rozgłaszanie zdarzeń, 128, 181, 182
rozsyłanie zdarzeń, 255
rozwój aplikacji komputerowych, 27
RPC, 316
Run, 114
rysowanie, 385

S

- sandbox, 132
- Scale content, 97
- Script, 110
- Scroller, 68, 414, 415
- SDK, 15
- seek(), 198
- sekcje produktów, 75
- selectedChild, 336
- selectedIndex, 247, 336
- selectedItem, 247
- selectionColor, 362, 363, 365, 366
- selektory, 366
 - selektor elementu, 366
 - selektor identyfikatorów, 375
 - selektor klasy, 366
 - selektor potomków, 374
- send(), 130, 134, 229, 232
- service-oriented architecture, 29
- set, 180, 241
- setFocus(), 104
- setStyle(), 364
- ShoppingCart, 162, 163, 164, 198
- ShoppingCartItem, 159, 164
 - refaktoryzacja, 202
- ShoppingList, 399, 404
- ShoppingView, 213, 218, 402, 406
 - DataGroup, 243
- siatki danych, 279
 - AdvancedDataGrid, 292
 - DataGrid, 280
 - dodawanie kontrolek edycji, 283
 - multipleSelection, 321
 - przeciągnij i upuść, 317
- Silverlight, 32, 33
- SkinnableComponent, 401, 406, 421
- SkinnableContainer, 66
- SkinnableDataContainer, 66
- SkinPart, 410, 414
- SkinState, 410, 411
- skórki, 66, 359, 360, 381, 410
 - Application, 393
 - ButtonSkin, 382
 - definiowanie wyglądu przycisku w różnych stanach, 389
 - klasa skórki, 384
 - komponenty, 401
 - komponenty Spark, 381
 - nazwy skórek, 382
 - skórka aplikacji, 393
 - stany komponentu, 385
 - tworzenie, 383, 393, 412
 - tworzenie renderera, 426
- SOA, 29
- SocialSecurityValidator, 349
- Software Development Kit, 15
- SolidColor, 389
- SolidColorStroke, 391
- Sort, 193, 195
- sortExpertMode, 292, 293, 294
- SortField, 193, 194, 195
- sortowanie, 292
 - kolekcje, 193
- Source, 124
- Spark, 280
- spark.components.Group, 213, 219, 339
- spark.components.TextInput, 240
- spark.layout.VerticalLayout, 339
- spark.layouts.HorizontalLayout, 213, 219
- społeczność Fleksa, 22
- sprawdzanie danych po stronie klienta, 349
- sprawdzanie poprawności danych, 347
- Sprite, 386
- stan zmiennej, 55
- stany komponentu, 385
- stany widoku, 81
 - currentState, 84
 - podstawowy stan, 81
 - sterowanie, 84
 - tworzenie, 81
- States, 81
- static, 156
- Step Into, 55, 58
- Step Over, 56, 59
- Step Return, 59
- sterowanie stanami widoku, 84
- stos, 336
- stosowanie
 - komponenty, 211
 - kontrolki, 93
 - style, 361
 - złożone kontenery, 88
- String, 109, 112, 123, 151, 152
- StringValidator, 349, 354
- strumień zdarzenia, 264, 265
 - faza docelowa, 265
 - faza propagacji, 265, 268
 - faza przechwytywania, 265
- style, 359
 - <mx:Style>, 365
 - color, 361
 - CSS, 359

dziedziczenie stylów, 364
fontFamily, 361
fontSize, 362
fontStyle, 362
fontWeight, 362
Label, 361
nadawanie stylów przez atrybuty znaczników, 362
nadawanie stylów za pomocą plików CSS, 368
nazwy stylów, 365
paddingLeft, 362
paddingRight, 362
pliki CSS, 368
przesłanie stylów we wczytanych plikach CSS, 379
pseudoselektory, 376
selektor elementu, 366
selektor identyfikatorów, 375
selektor klasy, 366
selektor potomków, 374
setStyle(), 364
stosowanie, 361
StyleManager, 379
textAlign, 362
textDecoration, 362
tworzenie pliku SWF z arkusza CSS, 378
wczytywanie arkusza stylów, 379
wczytywanie stylów, 378
zmiana stylów w trakcie działania aplikacji, 377
styleFunction, 296
StyleManager, 379
subclasses, 112
SUM, 306
SummaryField2, 309
summaryPlacement, 304, 310
SummaryRow, 304, 305, 309, 310
super(), 227, 230
SVG, 95
SWF, 49, 95
system klient-serwer, 28
system nawigacji, 335

T

tablice, 164
przetwarzanie, 166
target, 112, 115
technika przeciągnij i upuść, 313
technologie bogatych aplikacji internetowych, 32
technologie serwerowe platformy Flash, 37
text, 95, 99, 109, 175
textAlign, 362
TextArea, 94, 118

textDecoration, 362
TextInput, 94, 103, 240, 280, 352
TextInputDataRenderer, 246
TextInputRenderer, 246
this, 153, 161
tight coupling, 254
TileLayout, 67
toLowerCase(), 248
toString(), 153, 154, 161, 242
trace(), 154, 155, 158, 161, 162, 165, 215, 260, 264
Tree, 123
tryb edycji źródła, 42
tryb projektowania, 42, 72
tworzenie
 aplikacje MXML, 40, 103
 funkcja obsługi zdarzenia, 256
 klasy, 149, 406
 komponenty, 118, 209, 271, 399
 kursory, 198
 layout sklepu internetowego, 69
 metody dostępne, 241
 obiekt HTTPService, 129
 obiekty, 153, 156
 obiekty wartości, 150
 plik SWF z arkusza CSS, 378
 projekt, 39, 40
 renderer dla skórki, 426
 renderer elementów, 285, 307
 renderer elementów osadzony w kodzie, 287
 skórki, 383, 393, 412
 stany widoku, 81
 widok szczegółów, 98
 zarys layoutu w trybie edycji źródła, 70
 zdarzenia, 261
typy danych, 109

U

UIComponent, 208, 401
uint, 166
układanie kontrolerek, 93
ukrywanie funkcjonalności, 180
umieszczenie modelu poza aplikacją, 124
uniwersalny identyfikator zasobu, 44
unsigned integer, 160
upuszczanie, 314
URI, 44
uruchamianie aplikacji, 48
UserAcknowledgeEvent, 261
UserForm, 209
useVirtualLayout, 246, 247

usuwanie
 elementy kolekcji, 200
 projekt, 60
UUID, 60
użytkownicy końcowi, 31

V

validateAll(), 355
Validator, 349, 355
 validateAll(), 355
value object, 150
Value Object Example, 174
valueObjects, 151
var, 214
variableRowHeight, 286
Variables, 115, 142
VerticalLayout, 67, 69, 89, 247, 248, 272, 280, 339, 415
VGroup, 98, 99, 157, 214, 215
view states, 81
ViewStack, 66, 336, 337
visible, 82, 100, 158
void, 111

W

walidacja danych, 347
walidatory, 348, 353
waluty, 349
WarningDialog, 268
wartości logiczne, 109
wartości skalarne, 109
wartość null, 167
wczytywanie arkusza stylów, 379
wczytywanie stylów, 378
Web 2.0, 15
web-based, 28
wiązanie danych, 95, 101, 102, 118, 173, 184
 aktualizacja widoku, 174
 dwustronne wiązanie danych, 352
 kompilator, 179
 metody dostępne, 180
 zasada działania mechanizmu wiązania danych, 173
 zdarzenia, 182
widok, 212
 stany, 81
widok szczegółów, 98
wielokrotne wykorzystanie komponentów, 211
więzy, 76
Windows, 431
Windows Presentation Foundation, 34

wirtualizacja, 246
 implementacja, 246
 List, 247
wirtualna maszyna ActionScript, 36
wirtualna maszyna Javy, 33
właściwości pałapki, 59
WPF, 34
wprowadzanie danych, 94
wskazanie kursorem myszy, 98
wygląd aplikacji, 359
wykonywanie kodu krok po kroku, 53
wymagania systemowe, 21, 431
wypełnianie obszaru, 389
wyrażenia E4X, 140
wyskakujące okienka, 267
wysyłanie żądań, 130
wyszukiwanie danych, 197
wyświetlanie danych podsumowujących, 303
wyświetlanie danych podsumowujących
 za pomocą języka ActionScript, 309
wyświetlanie informacji o produkcie, 285
wyświetlanie obrazów, 95
wyświetlanie produktów, 271
wyświetlanie produktów z wybranej kategorii, 249
wyświetlanie przycisku Remove, 287
wyświetlanie testu, 372
wyświetlanie wartości produktów, 290
wytwarzanie oprogramowania oparte
 na komponentach, 207
wywołanie funkcji, 111
wzorzec MVC, 212

X

XAML, 32, 34
XML, 15, 50, 123, 184
XML (klasa), 135, 137
XMLDocument, 135
XMLList, 137, 138, 185, 190
XMLListCollection, 141, 143, 144, 145, 185, 189, 193, 227

Z

zachowanie stanu, 30
zamykanie znaczników, 51
zarys layoutu w trybie edycji źródła, 70
zarządzanie pobieraniem danych, 226
zasady tworzenia komponentów, 209
zastępczy obiekt przeciągania, 314
zbiory danych, 235
zdalne dane XML, 123

- zdarzenia, 107, 253, 417, 421
 - change, 250
 - clear, 109
 - click, 108, 109, 267
 - COLLECTION_CHANGE, 275
 - CollectionEvent, 276
 - creationComplete, 117, 118, 300, 302
 - dane obiektu zdarzenia, 111
 - dispatchEvent(), 183, 255
 - dragComplete, 318
 - dragDrop, 319
 - dragEnter, 319
 - dragExit, 319
 - dragOver, 319
 - Event, 112
 - faza propagacji, 264
 - funkcja nasłuchująca, 108
 - funkcja obsługi zdarzenia, 108
 - inspekcja obiektu zdarzenia, 114
 - klasy niestandardowych zdarzeń, 260
 - komponent rozgłaszający zdarzenie, 112
 - metadane, 259
 - mouseDown, 318, 323, 324
 - MouseEvent, 112, 253
 - mouseMove, 318, 324
 - mouseOver, 100
 - nasłuchiwaniec, 108
 - nasłuchiwanie, 182
 - nazwa zdarzenia, 112
 - nazwy funkcji obsługi zdarzeń, 114
 - obiekt docelowy, 265
 - obiekt zdarzenia, 111, 112
 - obsługa, 107
 - obsługa przez funkcję języka ActionScript, 110
 - podpowiadanie kodu, 109

- ProductEvent, 269, 274
- przekazywanie danych podczas wywoływania funkcji obsługi zdarzenia, 111
- result, 129, 265
- rozgłaszacz, 108
- rozgłaszanie, 182
- rozsyłanie zdarzeń, 255
- strumień zdarzenia, 264, 265
- tworzenie, 261
- UserAcknowledgeEvent, 261
- wiązanie danych, 182
- zdarzenia systemowe, 107, 117
- zdarzenia użytkownika, 107
- zestawy kontrolek, 280
- zintegrowane środowisko programistyczne, 36
- ZipCodeFormatter, 348
- ZipCodeValidator, 349, 353, 354
- zmiana
 - style w trakcie działania aplikacji, 377
 - wygląd aplikacji, 359
 - wygląd komponentów, 66
- zmienne, 55, 214
- znaczniki Flex, 44
- znaczniki FXG, 385, 386
- znaczniki MXML, 15, 68
 - właściwości, 68
 - znaczniki potomne, 68

Ż

- źródło przeciągania, 314

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 Flexie 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 działaniach wspomagających programistów i projektantów.

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



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



Helion

Nr katalogowy: 5853



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/newsletter>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 43

e-mail: helion@helion.pl

<http://helion.pl>

Flash Builder

Tworzenie interfejsu
użytkownika

Stosowanie kontrolek

Obsługa zdarzeń

ActionScript i MXML

Tworzenie
komponentów

Siatki danych

Rendering elementów

Stosowanie mecha-
nizmu „przeciągnij
i upuść”

Style i skórki

helion.pl
księgarnia
internetowa

Cena 77,00 zł

ISBN 978-83-246-2884-1



9 788324 628841

Informatyka w najlepszym wydaniu