

Wykład XI

Java - środowisko GUI Biblioteka JavaFX

Wymagania dotyczące aplikacji konsumenckich, a zwłaszcza aplikacji na urządzenia mobilne; wiążą się z bardzo dużą atrakcyjnością wizualną i funkcjonalnością interfejsu użytkownika. Do uzyskania lepszego wsparcia i uniwersalności tworzenie graficznych interfejsów użytkownika, konieczne było zastosowanie nowego rozwiązania i to właśnie ta potrzeba doprowadziła do powstania JavaFX. JavaFX to platforma kliencka oraz pakiet do tworzenia graficznego interfejsu użytkownika następnej generacji.

JavaFX udostępnia bardzo potężną, łatwą w użyciu i elastyczną platformę ułatwiającą tworzenie nowoczesnych i bardzo atrakcyjnych wizualnie interfejsów użytkownika.

W tym wykładzie celem będzie przedstawienie możliwości nowego pakietu oraz środowiska graficznego do tworzenia oprogramowania z wykorzystaniem platformy JavaFX.

JavaFX stanowi docelową platformę, która w przyszłości będzie używana do tworzenia graficznego interfejsu użytkownika aplikacji pisanych w Javie. Oczekuje się, że w ciągu kilku najbliższych lat JavaFX zastąpi Swing w nowych projektach.

Rozwój platformy JavaFX miał dwie główne fazy. Początkowo JavaFX bazowała na języku skryptowym o nazwie **JavaFX Script**. Ale przestał on być rozwijany. Zaczynając od wersji JavaFX 2.0, platforma JavaFX jest tworzona bezpośrednio w języku Java i udostępnia rozbudowane API. JavaFX obsługuje także język FXML, który może być używany do tworzenia interfejsu użytkownika.

Podstawowe pojęcia z zakresu JavaFX

Platforma JavaFX ma wszystkie dobre cechy pakietu Swing, została ona napisana bezpośrednio w Javie. Oprócz tego zapewnia wsparcie dla architektury MVC.

Przeważająca większość pojęć związanych z tworzeniem graficznego interfejsu użytkownika w pakiecie Swing odnosi się także do JavaFX. Niemniej jednak pomiędzy oboma rozwiązaniami występują znaczące różnice.

Z punktu widzenia programisty pierwszą różnicą pomiędzy JavaFX i pakietem Swing, którą można zauważyć, jest organizacja platformy JavaFX oraz wzajemne powiązania pomiędzy jej głównymi komponentami, JavaFX umożliwia tworzenie interfejsów użytkownika, które są bardziej dynamiczne pod względem wizualnym.

Pakiety JavaFX

Elementy JavaFX zostały umieszczone w pakietach, których nazwy rozpoczynają się od `javafx`. Biblioteka API platformy JavaFX składała się z ponad 30 pakietów. Oto cztery przykładowe z nich: `javafx.application`, `javafx.stage`, `javafx.scene` oraz `javafx.scene.layout`.

Klasy Stage oraz Scene

Podstawową elementem zaimplementowanym przez JavaFX jest **obszar roboczy** (ang. *scene*) zawierający **scenę**. Obszar roboczy definiuje przestrzeń, natomiast scena określa, co się w tej przestrzeni znajduje, inaczej: obszar roboczy jest kontenerem dla sceny, natomiast scena — dla elementów, które się na nią składają. W efekcie wszystkie aplikacje JavaFX zawierają przynajmniej jeden obszar roboczy oraz jedną scenę. W JavaFX elementy te są reprezentowane przez klasy `Stage` oraz `Scene`. W celu utworzenia aplikacji JavaFX należy w najprostszym przypadku dodać do obiektu `Stage` przynajmniej



jeden obiekt Scene. Przyjrzyjmy się nieco dokładniej tym dwóm klasom.

Klasa Stage jest pojemnikiem głównego poziomu. Wszystkie aplikacje JavaFX automatycznie mają dostęp do jednego obiektu Stage, nazywanego **głównym obszarem roboczym**. Ten główny obszar roboczy jest dostarczany przez środowisko wykonawcze podczas uruchamiania aplikacji JavaFX. Choć można także tworzyć inne obszary robocze, to w wielu aplikacjach ten główny będzie jedynym niezbędnym.

Zgodnie z podanymi wcześniej informacjami obiekt Scene jest pojemnikiem dla wszystkich elementów tworzących scenę. Elementami tymi mogą być różnego rodzaju kontrolki, takie jak przyciski, pola wyboru, pola tekstowe czy też obrazki. W celu utworzenia sceny wszystkie te elementy należy dodać do obiektu Scene.

Węzły i graf sceny

Poszczególne elementy sceny są nazywane **węzłami** (ang. *node*). Węzłem jest na przykład kontrolka przycisku. Jednak oprócz tego węzłami mogą także być grupy węzłów. Co więcej, węzeł może zawierać węzły potomne. W takim przypadku mówi się, że węzeł zawierający węzły potomne jest ich **rodzicem** lub **węzłem gałęzi** (ang. *branch node*). Z kolei węzły, które nie zawierają węzłów potomnych, są węzłami końcowymi, czyli tak zwanymi **liśćmi**. W grafie sceny istnieje jeden węzeł o szczególnym znaczeniu, jest to tak zwany **korzeń**. Jest to węzeł głównego poziomu i jako jedyny w całym grafie sceny nie ma rodzica. A zatem z wyjątkiem korzenia wszystkie inne węzły w grafie sceny mają węzeł rodzica i wszystkie, bezpośrednio lub pośrednio, są potomkami korzenia.

Klasą bazową wszystkich węzłów jest klasa Node. Dostępnych jest także kilka innych klas, które bezpośrednio lub pośrednio są podklasami klasy Node. Do tej grupy zaliczają się między innymi klasy: Parent, Group, Region oraz Control.

Układy

JavaFX udostępnia kilka paneli układów, które zarządzają procesem rozmieszczania elementów na scenie. Na przykład panel FlowPane tworzy układ rozmieszczający elementy jeden za drugim, a panel GridPane tworzy układ pozwalający rozmieszczać elementy w wierszach i kolumnach. Dostępnych jest także kilka innych układów, na przykład BorderPane (przypominający menedżera układu BorderLayout znanego z AWT). Wszystkie te panele układów są umieszczone w pakiecie javafx.scene.layout.

Klasa Application i metody cyklu życia

Każda aplikacja JavaFX musi być klasą dziedziczącą po klasie Application, umieszczoną w pakiecie javafx.application. Oznacza to, że klasy tworzonych przez nas aplikacji JavaFX będą rozszerzać klasę Application. Klasa ta definiuje trzy metody cyklu życia, które aplikacje mogą przesłaniać. Są nimi przedstawione poniżej metody: init(), start() oraz stop(). Kolejność ich wywoływania odpowiada kolejności, w jakiej zostały podane na poniższej liście:

```
void init()
```

```
abstract void start(Stage primaryStage)
```

```
void stop()
```

Metoda init() jest wywoływana podczas uruchamiania aplikacji. Służy ona do wykonywania różnych czynności związanych z inicjalizacją aplikacji. Klasy tej *nie można* używać ani do utworzenia obiektu obszaru roboczego, ani do skonstruowania sceny. Jeśli wykonywanie czynności inicjalizacyjnych nie jest konieczne, to metody tej nie trzeba przesłaniać, gdyż jest dostępna jej domyślna, pusta wersja.

Po metodzie init() wywoływana jest metoda start(). To od niej rozpoczyna się działanie aplikacji i to właśnie *jej* można używać do konstruowania sceny. Warto zauważyć, że do tej metody jest przekazywana referencja do obiektu Stage. Obiekt ten reprezentuje obszar roboczy dostarczony przez środowisko wykonawcze, czyli główny obszar roboczy.

Należy zauważyć, że start() jest metodą abstrakcyjną, co oznacza, że aplikacja musi ją przesłonić.

Podczas kończenia działania aplikacji wywoływana jest metoda stop(). Jest ona miejscem, w którym można wykonywać wszelkie czynności porządkowe oraz inne, związane z

zamykaniem aplikacji. Jeśli wykonywanie takich operacji nie jest potrzebne, metody stop() nie trzeba przesłaniać, gdyż jest dostępna jej domyślna, pusta wersja.

Uruchamianie aplikacji JavaFX

Aby uruchomić niezależną aplikację JavaFX, należy wywołać metodę launch() zdefiniowaną w klasie Application. Metoda ta ma dwie wersje; poniżej przedstawiona została ta, która będzie używana w tym rozdziale:

```
public static void launch(String ... args)
```

Parametr *args* reprezentuje listę łańcuchów, które zazwyczaj są argumentami podanymi w wierszu poleceń. Warto zwrócić uwagę, że lista ta może być pusta. W momencie wywołania metoda launch() powoduje utworzenie aplikacji oraz wywołanie najpierw metody init(), a następnie metody start(). Metoda launch() nie zwraca sterowania aż do momentu zakończenia aplikacji. Przedstawiona wcześniej wersja metody launch() uruchamia aplikację, czyli podklasę klasy Application, wewnątrz której metoda ta została wywołana. Druga wersja metody launch() pozwala określić inną klasę aplikacji, którą należy uruchomić.

Aplikacje JavaFX, które zostały spakowane przy użyciu programu narzędziowego *javafxpackager* (bądź analogicznych narzędzi udostępnianych przez zintegrowane środowiska programistyczne), nie muszą zawierać wywołania metody launch(). Niemniej jednak umieszczenie jawnego wywołania tej metody w kodzie aplikacji niejednokrotnie ułatwia proces jej testowania i uruchamiania, a oprócz tego zapewnia możliwość korzystania z programu bez umieszczania go w pliku JAR.

Szkielet aplikacji JavaFX

Wszystkie aplikacje JavaFX mają podobną podstawową strukturę. Dlatego zanim zajmiemy się innymi możliwościami platformy JavaFX, warto przedstawić bardzo prosty szkielet takiej aplikacji.

Oprócz przedstawienia podstawowej struktury aplikacji zamieszczony poniżej kod pokazuje także, w jaki sposób można ją uruchomić oraz jak wywoływać metody cyklu życia aplikacji. Komunikaty informujące o wywołaniach metod cyklu życia aplikacji są wyświetlane w oknie konsoli.

Przykładowy szkielet aplikacji JavaFX:

```
package application;

import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.layout.FlowPane;
import javafx.stage.*;

public class Okno_JavaFX extends Application {

    //Przesłonięta metoda init().
    public void init() {
        System.out.println("W metodzie init().");
    }

    //Przesłonięta metoda start().
    public void start(Stage myStage) {
        System.out.println("W metodzie start().");
        //Określa nazwę obszaru roboczego.
        myStage.setTitle("Szkielet aplikacji JavaFX");
        //Tworzy korzeń grafu sceny. W tym przypadku zostaje
        //użyty panel FlowPane, lecz dostępnych jest także
        //kilka alternatywnych paneli.
        FlowPane rootNode = new FlowPane();
        //Tworzy obiekt sceny.
        Scene myScene = new Scene(rootNode, 300, 200);
```

```

//Zapisuje scenę w obszarze roboczym.
myStage.setScene(myScene);
//Wyświetla obszar roboczy i scenę.
myStage.show();
}
//Przesłonięta wersja metody stop().
public void stop() {
    System.out.println("W metodzie stop().");
}

public static void main(String[] args) {
    System.out.println("Uruchamianie aplikacji JavaFX.");
    //Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
    launch(args);
}
}

```

Wątek aplikacji

Metoda `init()` nie może być używana do tworzenia obszaru roboczego ani sceny. Nie można ich także tworzyć w konstruktorze aplikacji. Wynika to z faktu, że obiekty te muszą być utworzone w **wątku aplikacji**. Okazuje się jednak, że zarówno konstruktor aplikacji, jak i metoda `init()` są wykonywane w **wątku uruchomieniowym**. Oznacza to, że nie można ich używać do tworzenia obiektów obszaru roboczego ani sceny. Zamiast tego, jak pokazano w przedstawionym wcześniej szkielecie aplikacji JavaFX, cały graficzny interfejs użytkownika aplikacji musi zostać skonstruowany i zainicjowany w metodzie `start()`, gdyż ona jest wywoływana w **wątku aplikacji**.

Wprowadzanie zmian do interfejsu użytkownika musi być wykonywane w wątku aplikacji. Zdarzenia realizowane w JavaFX są przekazywane do aplikacji w wątku aplikacji. Dzięki temu obiekty nasłuchujące mogą bez przeszkód manipulować graficznym interfejsem użytkownika aplikacji. Także metoda `stop()` jest wywoływana w wątku aplikacji.

Kontrolki JavaFX: Etykieta Label

Podstawowymi elementami wszystkich graficznych interfejsów użytkownika są kontrolki, gdyż to właśnie one zapewniają użytkownikom możliwość interakcji z aplikacjami. JavaFX udostępnia bogaty zbiór kontrolerek. Jedną z najprostszych z nich jest etykieta. Na przykładzie kontrolki etykiety przedstawiony zostanie proces tworzenia układu sceny. W JavaFX kontrolka etykiety jest obiektem klasy `Label`, która należy do pakietu `javafx.scene.control`. Klasa ta dziedziczy po klasie `Labeled`, która z kolei dziedziczy między innymi po klasie `Control`. Klasa `Labeled` definiuje kilka cech wspólnych dla wszystkich elementów, które mogą zawierać tekst; z kolei klasa `Control` definiuje cechy wspólne dla wszystkich kontrolerek.

Klasa `Label` definiuje trzy konstruktory; użyty zostanie następujący konstruktor:
`Label(String str)`

Parametr `str` określa łańcuch, który zostanie wyświetlony w etykiecie. Po utworzeniu etykiety (jak również dowolnej innej kontrolki) należy ją dodać do zawartości sceny, co jednocześnie oznacza dodanie jej do grafu sceny. W tym celu należy skorzystać z węzła korzenia używanego przez obiekt sceny i wywołać jego metodę `getChildren()`. Metoda ta zwraca listę węzłów potomnych, reprezentowaną w formie obiektu typu `ObservableList<Node>`. Interfejs `ObservableList` należy do pakietu `javafx.collections` i dziedziczy po interfejsie `java.util.List`, co oznacza, że udostępnia wszystkie możliwości list znanych z frameworku `Collections`. Używając listy węzłów potomnych zwróconej przez metodę `getChildren()`, można dodać do niej etykietę — w tym celu wystarczy wywołać metodę `add()` i przekazać do niej referencję do etykiety.

Aby usunąć kontrolkę z grafu sceny, należy wywołać metodę `remove()` interfejsu `ObservableList`. Oto przykład takiego wywołania:

```
rootNode.getChildren().remove(myLabel);
```

Powyższa instrukcja usuwa ze sceny komponent `myLabel`.

Poniższy program przedstawia przykład praktycznego zastosowania tych informacji, tworząc prostą aplikację JavaFX, która wyświetla etykietę:

```
package application;
```

```
// Prezentacja komponentu JavaFX - Label.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
public class Kontrolka_FX extends Application {

    public static void main(String[] args) {
// Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
launch(args);
}
// Przesłonięta metoda start().
public void start(Stage myStage) {
// Określa nazwę obszaru roboczego.
myStage.setTitle("Prezentacja etykiet JavaFX");
// Tworzy panel FlowPane, który stanie się węzłem korzenia.
FlowPane rootNode = new FlowPane();
// Tworzy obiekt sceny.
Scene myScene = new Scene(rootNode, 300, 200);
// Zapisuje scenę w obszarze roboczym.
myStage.setScene(myScene);
// Tworzy etykietę.
Label myLabel = new Label("To jest etykieta JavaFX.");
// Dodaje etykietę do grafu sceny.
rootNode.getChildren().add(myLabel);
// Wyświetla obszar roboczy i scenę.
myStage.show();
}
}
```

+

Kontrolki przycisków i obsługa zdarzeń

W poprzednim programie nie pokazano obsługi zdarzeń. Większość kontrolek graficznego interfejsu użytkownika generuje zdarzenia, które są obsługiwane przez kod aplikacji. Na przykład przyciski, pola wyboru oraz listy to jedne z wielu kontrolek, które w przypadku użycia generują zdarzenia. Obsługa zdarzeń w aplikacjach JavaFX przypomina obsługę zdarzeń w aplikacjach korzystających z pakietów Swing lub AWT, a nawet jest nieco łatwiejsza.

Jedną z najczęściej używanych kontrolek są przyciski. Wynika z tego, że zdarzenia generowane przez przyciski także są jednymi z najczęściej obsługiwanych.

Właśnie dlatego prezentacja przycisków oraz obsługa zdarzeń zostały połączone i opisane wspólnie.

Podstawowe informacje o zdarzeniach

W JavaFX klasą bazową wszystkich zdarzeń jest klasa `Event`, umieszczona w pakiecie `javafx.event`.

Klasa ta dziedziczy po `java.util.EventObject`, co oznacza, że wszystkie zdarzenia JavaFX dysponują tymi samymi podstawowymi możliwościami co pozostałe zdarzenia w języku Java. Dostępnych jest kilka klas potomnych klasy `Event`. Jedną z nich, a jednocześnie tą, która zostanie przedstawiona w kolejnym przykładzie, jest klasa `ActionEvent`. Obsługuje ona zdarzenia generowane przez przyciski.

Ogólnie rzecz ujmując, platforma JavaFX obsługuje zdarzenia, korzystając z modelu delegacji.

A zatem aby obsłużyć zdarzenie, w pierwszej kolejności należy zarejestrować obiekt nasłuchujący, który będzie na nie oczekiwał. W momencie wystąpienia zdarzenia zostanie wywołany odpowiedni obiekt nasłuchujący. Musi on obsłużyć zdarzenie, a następnie oddać sterowanie. Jak zatem widać, zdarzenia JavaFX są obsługiwane bardzo podobnie do zdarzeń stosowanych w pakiecie Swing.

Zdarzenia są obsługiwane poprzez zaimplementowanie interfejsu `EventHandler`, zdefiniowanego w pakiecie `javafx.event`. Jest to interfejs sparametryzowany mający następującą deklarację:

```
interface EventHandler<T> extends Event<T>
```

Parametr typu `T` określa typ zdarzeń, które będą obsługiwane. Interfejs ten definiuje tylko jedną metodę, `handle()`, której parametrem jest obiekt zdarzenia. Postać tej metody została przedstawiona poniżej:

```
void handle(T eventObj)
```

Parametr `eventObj` reprezentuje wygenerowane zdarzenie. Obiekty nasłuchujące są zazwyczaj implementowane przy użyciu anonimowych klas wewnętrznych, choć można w tym celu używać także niezależnych klas, o ile tylko takie rozwiązanie będzie uzasadnione (na przykład jeśli ten sam obiekt nasłuchujący będzie obsługiwał zdarzenia generowane przez kilka źródeł).

Istnieje możliwość określenia źródła, które wygenerowało zdarzenie. W szczególności dotyczy to tych sytuacji, gdy ten sam obiekt nasłuchujący jest używany do obsługi zdarzeń pochodzących z kilku różnych źródeł. Źródło zdarzenia można określić, wywołując przedstawioną poniżej metodę `getSource()`, dziedziczoną po klasie `java.util.EventObject`:

```
Object getSource()
```

Inne metody klasy `Event` pozwalają określić typ zdarzenia, sprawdzić, czy zdarzenie zostało przetworzone, przetworzyć zdarzenie, wygenerować zdarzenie oraz określić jego element docelowy.

Kiedy zdarzenie zostanie przetworzone, nie jest już ono przekazywane do nadrzędnego obiektu nasłuchującego.

W platformie JavaFX zdarzenia są przetwarzane przy wykorzystaniu **łańcucha przydzielania zdarzeń**. Po wygenerowaniu zdarzenia jest ono przekazywane do węzła korzenia, stanowiącego początek tego łańcucha. Następnie zdarzenie jest przekazywane w

dół łańcucha, aż do elementu docelowego. Kiedy węzeł docelowy obsłuży zdarzenie, jest ono przekazywane z powrotem aż na początek łańcucha, dzięki czemu, jeśli pojawi się taka konieczność, będzie ono mogło zostać obsłużone przez węzły przodków elementu docelowego. Rozwiązanie to nosi nazwę **propagacji zdarzeń**. Jakiś węzeł należący do łańcucha może przetworzyć zdarzenie, co sprawi, że nie będzie już ono dalej przekazywane.

Kontrolka - Button

W platformie JavaFX kontrolka przycisku jest reprezentowana przez klasę Button, zdefiniowaną w pakiecie javafx.scene.control. Klasa Button dziedziczy po stosunkowo wielu klasach bazowych, do których należą między innymi ButtonBase, Labeled, Region, Control oraz Node.

Przyciski JavaFX mogą zawierać tekst, grafikę bądź oba te elementy.

Klasa Button definiuje trzy konstruktory. Podstawowy, ma następującą postać:

Button(String str)

W tym przypadku str jest łańcuchem określającym komunikat wyświetlany na przycisku. Kliknięcie przycisku powoduje wygenerowanie zdarzenia ActionEvent. ActionEvent jest klasą zdefiniowaną w pakiecie javafx.event. Obiekt nasłuchujący obsługujący zdarzenia tego typu można zarejestrować, wywołując metodę setOnAction(), której ogólna postać została przedstawiona poniżej:

final void setOnAction(EventHandler<ActionEvent> handler)

Parametr handler jest rejestrowanym obiektem nasłuchującym. Jak już wspomniano, obiektami nasłuchującymi są przeważnie anonimowe klasy wewnętrzne lub wyrażenia lambda. Metoda setOnAction() ustawia wartość właściwości onAction, która przechowuje referencję do obiektu nasłuchującego.

Podobnie jak w przypadku wcześniejszych technologii związanych z obsługą zdarzeń w języku Java, także i teraz obiekty nasłuchujące powinny obsługiwać zdarzenia tak szybko, jak to tylko możliwe, a następnie oddawać sterowanie. Jeśli obsługa zdarzeń przez obiekt nasłuchujący zajmuje zbyt wiele czasu, może to doprowadzić do zauważalnego spadku wydajności działania aplikacji. A zatem w przypadku długotrwałych operacji należy je realizować w odrębnych wątkach.

Przykład zastosowania przycisku i obsługi zdarzeń:

```
package application;
// Prezentacja zdarzeń i przycisków JavaFX.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
public class Przycisk_FX extends Application {
    Label response;
    public static void main(String[] args) {
        // Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
        Launch(args);
    }
    // Przesłonięta metoda start().
    public void start(Stage myStage) {
        // Określa nazwę obszaru roboczego.
        myStage.setTitle("Prezentacja przycisków i zdarzeń JavaFX");
        // Tworzy panel FlowPane, który stanie się węzłem korzenia.
        // W tym przypadku pionowe i poziome odstępki pomiędzy
        // komponentami będą miały wielkość 10.
        FlowPane rootNode = new FlowPane(20,20);
        // Wyrównuje kontrolki na scenie do środka.
        rootNode.setAlignment(Pos.CENTER);
        // Tworzy obiekt sceny.
        Scene myScene = new Scene(rootNode, 300, 100);
```

```
// Zapisuje obiekt sceny w obszarze roboczym.
myStage.setScene(myScene);
// Tworzy etykietę.
response = new Label("Kliknij przycisk");
// Tworzy dwa przyciski.
Button btnAlpha = new Button("Alfa");
Button btnBeta = new Button("Beta");
// Obsługa zdarzeń ActionEvent przycisku Alfa.
btnAlpha.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent ae) {
        response.setText("Kliknięto przycisk Alfa.");
    }
});
// Obsługa zdarzeń ActionEvent przycisku Beta.
btnBeta.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent ae) {
        response.setText("Kliknięto przycisk Beta.");
    }
});
//Dodaje etykietę i przyciski do grafu sceny.
rootNode.getChildren().addAll(btnAlpha, btnBeta, response);
//Wyświetla obszar roboczy i scenę.
myStage.show();
}
```

Kontrolki Image i ImageView

Kilka kontrolek JavaFX pozwala na wyświetlanie wewnątrz nich obrazów. Przykładami takich kontrolek są etykiety i przyciski, które pozwalają na wyświetlanie obrazów oraz tekstu. Oprócz tego obrazy można także dodawać bezpośrednio na scenie. Podstawowe możliwości obsługi obrazów w JavaFX zapewniają dwie klasy: Image oraz ImageView. Pierwsza z nich hermetyzuje sam obrazek jako taki, z kolei klasa ImageView udostępnia możliwości niezbędne do jego wyświetlania. Obie te klasy należą do pakietu `javafx.scene.image`.

Klasa Image wczytuje obraz ze strumienia `InputStream`, wskazanego adresu URL bądź ścieżki do pliku graficznego. Definiuje ona kilka konstruktorów, w tym poniższy:

`Image(String url)`

Parametr *url* określa adres URL lub ścieżkę dostępu do pliku graficznego. Jeśli przekazany argument nie będzie prawidłowo zapisanym adresem URL, to klasa przyjmie, że stanowi on ścieżkę do pliku. Jeśli okaże się, że argument jest prawidłowym adresem URL, to obraz zostanie pobrany.

Pozostałe konstruktory klasy Image pozwalają na określanie różnego rodzaju opcji, takich jak szerokość i wysokość obrazu. I jeszcze jedna uwaga: klasa Image nie dziedziczy po klasie Node, co oznacza, że jej instancji nie można dodawać do grafu sceny.

W celu wyświetlania obrazu po utworzeniu obiektu Image należy utworzyć obiekt ImageView. Klasa ImageView dziedziczy po Node, co oznacza, że jej obiekty mogą być dodawane do grafu sceny. Klasa ta definiuje trzy konstruktory. Poniżej został przedstawiony pierwszy z nich:

`ImageView(Image image)`

Tworzy on obiekt ImageView, który będzie prezentował obraz określony przez przekazany obiekt Image.

Przykładowy kod programu:

```
package application;
```

```
// Program wczytuje i wyświetla obraz.
import javafx.application.*;
```




```

import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.geometry.*;
import javafx.scene.image.*;
public class Obraz_FX extends Application {
public static void main(String[] args) {
// Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
launch(args);
}
// Przesłonięta metoda start().
public void start(Stage myStage) {
// Określa tytuł obszaru roboczego.
myStage.setTitle("Wyświetlanie obrazu");
// Tworzy panel FlowPane, który zostanie użyty jako węzeł korzenia.
FlowPane rootNode = new FlowPane();
// Wyrównuje kontrolki na scenie do środka.
rootNode.setAlignment(Pos.CENTER);
// Tworzy obiekt sceny.
Scene myScene = new Scene(rootNode, 300, 200);
// Zapisuje obiekt sceny w obszarze roboczym.
myStage.setScene(myScene);
// Tworzy obraz.
Image zima = new Image("zima.png");
//Image zima = new Image("http://img-
fotki.yandex.ru/get/4613/41885099.223/0_69ee5_3061cc7c_L.jpg");
// Tworzy obiekt ImageView korzystający z utworzonego
// wcześniej obiektu Image.
ImageView zima_View = new ImageView(zima);
//Dodaje obraz do grafu sceny.
rootNode.getChildren().add(zima_View);
//Wyświetla obszar roboczy i scenę.
myStage.show();
}
}

```

Umieszczanie obrazów w innych kontrolkach Dodawanie obrazów do etykiet

Zgodnie z informacjami podanymi w poprzednim rozdziale klasa Label reprezentuje etykiety. Taka etykieta może prezentować łańcuch, grafikę lub jedno i drugie.

Należy skorzystać z poniższego konstruktora klasy Label:

Label(String *str*, Node *image*)

Parametr *str* określa komunikat tekstowy, natomiast parametr *image* — obraz, który zostanie wyświetlony w etykiecie. Warto zwrócić uwagę, że parametr *image* jest typu Node. Takie rozwiązanie zapewnia bardzo dużą elastyczność, jeśli chodzi o obrazy dodawane do etykiet, jednak do naszych celów wystarczy użycie obiektu ImageView.

Przykładowy kod programu:

```

package application;
//Stosowanie obrazów w etykietach.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.geometry.*;
import javafx.scene.image.*;
public class Etykieta_FX extends Application {

```



```

public static void main(String[] args) {
    //Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
    launch(args);
}
//Przesłonięta metoda start().
public void start(Stage myStage) {
    //Określa nazwę obszaru roboczego.
    myStage.setTitle("Stosowanie obrazów w etykietach");
    //Tworzy panel FlowPane, który zostanie użyty jako węzeł korzenia.
    FlowPane rootNode = new FlowPane();
    //Wyrównuje kontrolki na scenie do środka.
    rootNode.setAlignment(Pos.CENTER);
    //Tworzy obiekt sceny.
    Scene myScene = new Scene(rootNode, 600, 500);
    //Zapisuje obiekt sceny w obszarze roboczym.
    myStage.setScene(myScene);
    //Tworzy obiekt ImageView zawierający wskazany obraz.
    ImageView zima_View = new ImageView("zima.png");

    //Tworzy etykietę zawierającą tekst i obraz.
    Label zima_Label = new Label("Zima", zima_View);
    //ustawienie sposobu wyświetlania obrazu i tekstu
    zima_Label.setContentDisplay(ContentDisplay.TOP);
    //Dodaje etykietę do grafu sceny.
    rootNode.getChildren().add(zima_Label);
    //Wyświetla obszar roboczy i scenę.
    myStage.show();
}
}

```

Wyświetlony został zarówno tekst, jak i obraz. Warto zwrócić uwagę, że tekst został umieszczony z prawej strony obrazu. To domyślny sposób działania etykiet; można go zmienić, wywołując metodę `setContentDisplay()`. Poniżej przedstawiona została deklaracja tej metody:

```
final void setContentDisplay(ContentDisplay position)
```

Wartość parametru *position* określa, w jaki sposób zostaną wyświetlone tekst i obraz.

Musi to być jedna ze stałych zdefiniowanych w typie wyliczeniowym `ContentDisplay`:

- BOTTOM RIGHT
- CENTER TEXT_ONLY
- GRAPHICS_ONLY TOP
- LEFT

Dodawanie obrazów w przyciskach

W JavaFX przyciski są reprezentowane przez klasę `Button`.

Przyciski oprócz tekstu mogą zawierać grafikę lub tekst i grafikę w zależności od zastosowanego konstruktora.

W pierwszej kolejności należy utworzyć obiekt `ImageView` prezentujący wybrany obraz, a następnie dodać go do przycisku.

Jednym ze sposobów, by to zrobić, jest skorzystanie z poniższego konstruktora:

```
Button(String str, Node image)
```

Parametr *str* określa tekst prezentowany na przycisku, a parametr *image* — obraz, który zostanie na nim wyświetlony. Położenie obrazu w stosunku do tekstu na przycisku można określić w taki sam sposób jak w przypadku etykiet — przy użyciu metody `setContentDisplay()`.

Przykład aplikacji:

```

package application;

//Umieszczanie obrazów na przyciskach.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
import javafx.scene.image.*;
public class Przycisk_FX extends Application {
    Label response;
    public static void main(String[] args) {
        //Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
        launch(args);
    }
    //Przesłonięta metoda start().
    public void start(Stage myStage) {
        //Określa nazwę obszaru roboczego.
        myStage.setTitle("Umieszczanie obrazów na przyciskach");
        //Tworzy panel FlowPane, który zostanie użyty jako węzeł korzenia. W tym
        //przypadku poziome i pionowe odstęp między kontrolkami wynoszą 10.
        FlowPane rootNode = new FlowPane(10, 10);
        //Wyrównuje kontrolki na scenie do środka.
        rootNode.setAlignment(Pos.CENTER);
        //Tworzy obiekt sceny.
        Scene myScene = new Scene(rootNode, 250, 450);
        //Zapisuje obiekt sceny w obszarze roboczym.
        myStage.setScene(myScene);
        //Tworzy etykietę.
        response = new Label("Kliknij przycisk");
        //Tworzy dwa przyciski z tekstem i obrazkiem.
        Button btnzima = new Button("Zima",
            new ImageView("zima.png"));
        Button btnlato = new Button("Lato", new ImageView("http://www.na-
            pulpit.com/tapety/lato-zagajnik-brzozy-rzeka-kladka.jpeg"));
        //Button btnlato = new Button("Lato", new ImageView("lato.png"));
        //Button btnlato = new Button("Lato",
            new ImageView("lato.png"));
        //Rozmieszcza tekst poniżej obrazu.
        btnzima.setContentDisplay(ContentDisplay.TOP);
        btnlato.setContentDisplay(ContentDisplay.TOP);
        //Obsługa zdarzeń przycisku zima.
        btnzima.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Kliknięto zima.");
            }
        });
        //Obsługa zdarzeń przycisku lato.
        btnlato.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Kliknięto lato.");
            }
        });
        //Dodaje etykietę i przyciski do grafu sceny.
        rootNode.getChildren().addAll(btnzima, btnlato, response);
        //Wyświetla obszar roboczy i scenę.
        myStage.show();
    }
}

```



Kontrolka RadioButton

Kolejnym rodzajem przycisków udostępnianych przez JavaFX są **przyciski opcji**. Przyciski opcji to grupy wzajemnie wykluczających się przycisków, z których w danej chwili tylko jeden może być zaznaczony. Przyciski opcji są reprezentowane przez klasę `RadioButton`, która dziedziczy zarówno po klasie `ButtonBase`, jak i `ToggleButton`. Oprócz tego klasa ta implementuje także interfejs `Toggle`.

Do tworzenia kontrolki `RadioButton` będzie używany następujący konstruktor:

```
RadioButton(String str)
```

Parametr *str* określa etykietę przycisku. Podobnie jak w przypadku wszystkich innych przycisków kliknięcie przycisku opcji powoduje wygenerowanie zdarzenia.

Ze względu na swój charakter przyciski opcji muszą być organizowane w grupy. W dowolnej chwili tylko jeden przycisk w danej grupie może być zaznaczony. Jeśli użytkownik kliknie jeden z przycisków opcji należących do grupy, to wybrany dotychczas przycisk przestanie być zaznaczony. Grupy przycisków opcji są tworzone przy użyciu klasy `ToggleGroup`, należącej do pakietu `javafx.scene.control`.

Klasa ta udostępnia jedynie domyślny konstruktor.

Przyciski opcji są dodawane do grupy poprzez wywołanie na rzecz kontrolki przycisku metody `setToggleGroup()` zdefiniowanej w klasie `ToggleButton`. Jej deklaracja została przedstawiona poniżej:

```
final void setToggleGroup(ToggleGroup tg)
```

Parametr *tg* jest referencją do obiektu `ToggleGroup` reprezentującego grupę, do której dany przycisk opcji ma należeć. Po dodaniu do grupy wszystkich przycisków opcji zyskają one charakterystyczne dla siebie działanie. Ogólnie rzecz biorąc, jeśli przyciski opcji są stosowane w grupach, to podczas początkowego wyświetlania interfejsu użytkownika aplikacji jeden z tych przycisków będzie zaznaczony. Poniżej opisane zostały dwa sposoby, jak określić ten początkowo zaznaczony przycisk. Pierwszym rozwiązaniem jest wywołanie metody `setSelected()` na rzecz przycisku, który należy zaznaczyć. Metoda ta, przedstawiona poniżej, jest zdefiniowana w klasie `ToggleButton` (jednej z klas bazowych klasy `RadioButton`):

```
final void setSelected(boolean state)
```

Jeśli parametr *state* przyjmie wartość `true`, to przycisk opcji zostanie zaznaczony; w przeciwnym razie zaznaczenie zostanie z niego usunięte. W przypadku użycia tej metody nie będzie generowane żadne zdarzenie, nawet jeśli przycisk opcji zostanie zaznaczony. Drugim sposobem początkowego zaznaczenia przycisku opcji jest wywołanie na jego rzecz metody `fire()`. Poniżej przedstawiona została jej deklaracja:

```
void fire()
```

Wywołanie tej metody powoduje wygenerowanie zdarzenia skierowanego do przycisku, o ile nie był on wcześniej zaznaczony. Najprostszym z nich jest odpowiadanie na zdarzenia `ActionEvent`, generowane w momencie zaznaczania przycisków.

+

```

package application;
// Prosty przykład stosowania przycisków opcji.
//
// Ten program odpowiada na zdarzenia ActionEvent
// generowane przez wybierane przyciski opcji. Program
// pokazuje także, jak można programowo generować
// zdarzenia.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
public class Przycisk_Opcji_FX extends Application {
    Label response;
    public static void main(String[] args) {
        // Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
        launch(args);
    }
    // Przesłonięta metoda start().
    public void start(Stage myStage) {
        // Określa nazwę obszaru roboczego.
        myStage.setTitle("Demonstracja działania Radio Buttons");
        // Tworzy panel FlowPane, który zostanie użyty jako węzeł korzenia. W tym
        // przypadku odstęp między komponentami wynosi 10.
        FlowPane rootNode = new FlowPane(10, 10);
        // Wyrównuje kontrolki na scenie do środka.
        rootNode.setAlignment(Pos.CENTER);
        // Tworzy obiekt sceny.
        Scene myScene = new Scene(rootNode, 290, 120);
        // Zapisuje obiekt sceny w obszarze roboczym.
        myStage.setScene(myScene);
        // Tworzy etykietę prezentującą aktualnie wybraną opcję.
        response = new Label("");
        // Tworzy przyciski opcji.
        RadioButton rbTrain = new RadioButton("Pociąg");
        RadioButton rbCar = new RadioButton("Samochód");
        RadioButton rbPlane = new RadioButton("Samolot");
        // Tworzy grupę przycisków opcji.
        ToggleGroup tg = new ToggleGroup();
        // Dodaje poszczególne przyciski do grupy.
        rbTrain.setToggleGroup(tg);
        rbCar.setToggleGroup(tg);
        rbPlane.setToggleGroup(tg);
        // Obiekt nasłuchujący obsługujący zdarzenia generowane przez przyciski opcji.
        rbTrain.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Wybrany środkiem transportu jest pociąg.");
            }
        });
        rbCar.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Wybrany środkiem transportu jest samochód.");
            }
        });
        rbPlane.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent ae) {
                response.setText("Wybrany środkiem transportu jest samolot.");
            }
        });
        // Generuje zdarzenie dla pierwszego przycisku opcji.
        // Powoduje to zaznaczenie przycisku i wygenerowanie
        // zdarzenia ActionEvent.

```



```

rbTrain.fire();
// Dodaje etykietę i przyciski do grafu sceny.
rootNode.getChildren().addAll(rbTrain, rbCar, rbPlane, response);
// Wyświetla obszar roboczy i scenę.
myStage.show();
}
}

```

Obsługa zdarzeń w grupie

Jedną z metod obsługi zdarzeń jest `ActionEvent`, to jednak czasami lepszym rozwiązaniem może być obsługa zdarzeń informujących o zmianach stanu grupy. Kiedy takie zdarzenie zostanie zgłoszone, obsługujący je obiekt nasłuchujący bez trudu będzie mógł określić, który przycisk opcji został zaznaczony, i odpowiednio zareagować. Aby skorzystać z tego rozwiązania, należy zarejestrować obiekt nasłuchujący obsługujący zdarzenia typu `ChangeListener` w obiekcie `ToggleGroup`.

Przykład kodu:

```

package application;
// Prosty przykład stosowania przycisków opcji.
//
// Ten program odpowiada na zdarzenia ActionEvent
// generowane przez wybierane przyciski opcji. Program
// pokazuje także, jak można programowo generować
// zdarzenia.
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
import javafx.beans.value.*;

public class Przycisk_Opcji_FX extends Application {
    Label response;
    public static void main(String[] args) {
        // Wywołuje metodę launch(), która uruchamia aplikację JavaFX.
        launch(args);
    }
    // Przesłonięta metoda start().
    public void start(Stage myStage) {
        // Określa nazwę obszaru roboczego.
        myStage.setTitle("Demonstracja działania Radio Buttons");
        // Tworzy panel FlowPane, który zostanie użyty jako węzeł korzenia. W tym
        // przypadku odstęp między komponentami wynosi 10.
        FlowPane rootNode = new FlowPane(10, 10);
        // Wyrównuje kontrolki na scenie do środka.
        rootNode.setAlignment(Pos.CENTER);
        // Tworzy obiekt sceny.
        Scene myScene = new Scene(rootNode, 290, 120);
        // Zapisuje obiekt sceny w obszarze roboczym.
        myStage.setScene(myScene);
        // Tworzy etykietę prezentującą aktualnie wybraną opcję.
        response = new Label("");
        // Tworzy przyciski opcji.
        RadioButton rbTrain = new RadioButton("Pociąg");
        RadioButton rbCar = new RadioButton("Samochód");
        RadioButton rbPlane = new RadioButton("Samolot");
        // Tworzy grupę przycisków opcji.
        ToggleGroup tg = new ToggleGroup();
        // Dodaje poszczególne przyciski do grupy.
        rbTrain.setToggleGroup(tg);

```

```

rbCar.setToggleGroup(tg);
rbPlane.setToggleGroup(tg);
//Używa obiektu nasłuchującego, by odpowiadać na zdarzenia
//zmiany stanu grupy przycisków opcji.
tg.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
    public void changed(ObservableValue<? extends Toggle> changed,
        Toggle oldVal, Toggle newVal) {
        //Rzutuje nową zaznaczoną kontrolkę do typu RadioButton.
        RadioButton rb = (RadioButton) newVal;
        //Wyświetla komunikat o zaznaczonym przycisku.
        response.setText("Wybrany środek transportu to: " + rb.getText());
    }
});
//Zaznacza pierwszy przycisk opcji. Spowoduje to wygenerowanie
//zdarzenia informującego o zmianie stanu grupy.
rbTrain.setSelected(true);
// Dodaje etykietę i przyciski do grafu sceny.
rootNode.getChildren().addAll(rbTrain, rbCar, rbPlane, response);
// Wyświetla obszar roboczy i scenę.
myStage.show();
}
}

```

Opracowano na podstawie:

HERBERT SCHILDT - „Java. Kompendium programisty. Wydanie IX” Wydawnictwo Helion.

+