JavaFX

Zaawansowane technologie Javy 2019

Historia postania JavaFX

- Początkowym pakietem do tworzenia aplikacji GUI w Javie był pakiet AWT.
- Niedługo po wprowadzeniu został on zastąpiony pakietem Swing, który udostępnił znacznie lepsze rozwiązania. Choć pakiet Swing odniósł ogromny sukces, to jednak dosyć trudno w nim tworzyć „wizualne cuda”.
- W końcowyn efekcie powstał system JavaFX — platforma do tworzenia graficznych interfejsów użytkownika nowej generacji. JavaFX dostarczona została wraz z JDK 7.

Pakiety JavaFX

- Cała zawartość platformy JavaFX została umieszczona w pakietach, których nazwy zaczynają się od `javafx`.
- Zaczynając od JDK 9, pakiety JavaFX zostały zorganizowane w formie modułów `javafx.base`, `javafx.graphics` oraz `javafx.controls`.

Klasy Stage oraz Scene

- Główną metaforą zaimplementowaną w JavaFX jest obszar roboczy (ang. stage). W przypadku prawdziwych sztuk scenicznych obszar roboczy zawiera scenę (ang. scene).
- A zatem potocznie mówiąc: obszar roboczy definiuje przestrzeń, a scena określa, co się w tej przestrzeni pojawi.
- Obiekt `Stage` jest pojemnikiem najwyższego poziomu. Wszystkie aplikacje JavaFX automatycznie mają dostęp do jednego obiektu `Stage`, nazywanego głównym obszarem roboczym (ang. Primary stage).
- Obiekt `Scene` jest pojemnikiem dla wszystkich elementów, które składają się na scenę (są nimi kontrolki, takie jak przyciski, pola wyboru, pola tekstowe oraz inne elementy graficzne). W celu utworzenia sceny dodać należy wszystkie te elementy do obiektu klasy `Scene`.

Węzły i graf sceny

- Pojedyncze elementy sceny są nazywane węzłami (ang. node). Na przykład takim węzłem będzie kontrolka przycisku.
- Węzły mogą także zawierać grupy innych węzłów. Węzeł może także zawierać inny węzeł potomny. W takim przypadku węzeł zawierający jakieś dziecko jest nazywany węzłem rodzica (ang. parent node) lub węzłem gałęzi (ang. branch mode).
- Z kolei węzły, które nie zawierają węzłów potomnych, są nazywane węzłami końcowymi (ang. terminal node) lub liśćmi.

Węzły i graf sceny

- Kolekcja wszystkich węzłów tworzących scenę jest określana jako graf sceny (ang. scene graph) i stanowi drzewo.
- W grafie sceny występuje jeden, specjalny typ węzła — korzeń (ang. root node). Jest to najwyższy węzeł w grafie sceny i jednocześnie jedyny, który nie ma żadnego rodzica. A zatem wszystkie inne węzły należące do grafu sceny mają rodzica i wszystkie bezpośrednio lub pośrednio są potomkami korzenia.
- Klasą bazową dla wszystkich węzłów jest `Node`.
- Istnieje kilka różnych klas, które bezpośrednio lub pośrednio są jej klasami pochodnymi; między innymi należą do nich: `Parent`, `Group`, `Region` oraz `Control`.

Układy

- JavaFX udostępnia kilka paneli układu obsługujących proces rozmieszczania innych elementów na scenie.
- Na przykład klasa `FlowPane` tworzy układ rozmieszczający elementy jeden za drugim, a klasa `GridPane` — układ pozwalający na umieszczanie elementów w wierszach i kolumnach. Dostępnych jest także kilka innych układów, takich jak `BorderPane`. Każdy z nich dziedziczy po klasie `Node`.
- Wszystkie klasy układów zostały umieszczone w pakiecie `javafx.scene.layout`.

Klasa Application oraz metody cyklu życia

- Aplikacja JavaFX musi być klasą pochodną klasy `Application` zdefiniowanej w pakiecie `javafx.application`.
- Klasa ta definiuje trzy metody cyklu życia, które aplikacja może przesłaniać: `init()`, `start()` oraz `stop()`.
- Metoda `init()` jest wywoływana na samym początku działania aplikacji. Służy ona do wykonywania różnych czynności inicjalizacyjnych. Niemniej jednak nie można jej używać do tworzenia obiektu sceny ani do jej konstruowania.
- Jeśli aplikacja nie potrzebuje żadnych czynności inicjalizacyjnych, to metody `init()` nie trzeba przesłaniać, gdyż istnieje jej wersja domyślna.

Klasa Application oraz metody cyklu życia

- Po metodzie `init()` wywoływana jest metoda `start()` – stanowi ona początek działania aplikacji i może posłużyć do skonstruowania i przygotowania sceny.
- Parametrem tej metody jest obiekt klasy `Scene` dostarczony przez środowisko uruchomieniowe i stanowi scenę główną aplikacji.
- Jest to metoda abstrakcyjna i aplikacja musi ją nadpisać.

Klasa Application oraz metody cyklu życia

- Kiedy aplikacja kończy działanie, wywoływana jest z kolei metoda `stop()`, która pozwala na wykonanie wszelkich czynności porządkowych związanych z zamykaniem aplikacji.
- W przypadkach, gdy wykonywanie takich czynności nie jest potrzebne, można skorzystać z pustej, domyślnej wersji tej metody.

Uruchamianie aplikacji JavaFX

- Aby uruchomić niezależną aplikację JavaFX, należy wywołać metodę `launch()` klasy `Application`:
  `public static void launch(String ... args)`
- Parametr `args` reprezentuje listę (być może pustą) łańcuchów znakowych, które zazwyczaj będą określały argumenty wiersza poleceń.
- Wywołanie metody `launch()` powoduje utworzenie aplikacji, a następnie wywołanie jej metod `init()` i `start()`.
- Wywołanie metody `launch()` zakończy się dopiero po zakończeniu aplikacji.

Szkielet aplikacji JavaFX

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;

public class JavaFXSkel extends Application {
  public static void main(String[] args) {
    System.out.println("Uruchamiamy aplikację JavaFX.");
    // Uruchamia aplikację JavaFX,
    // wywołując metodę launch().
    launch(args);
  }
```

Szkielet aplikacji JavaFX

```
// Przesłonięcie metody init().
  public void init() {
    System.out.println("Wewnątrz metody init().");
  }
  // Przesłania metodę stop().
  public void stop() {
    System.out.println("Wewnątrz metody stop().");
  }
```

Szkielet aplikacji JavaFX

```
  // Przesłonięcie metody start().
  public void start(Stage myStage) {
    System.out.println("Wewnątrz metody start().");
    // Określa tytuł sceny.
    myStage.setTitle("Szkielet aplikacji JavaFX.");
    // Tworzy korzeń. W tym przypadku będzie to układ
    // FlowPane, choć istnieje także kilka innych.
    FlowPane rootNode = new FlowPane();
    // Tworzy scenę.
    Scene myScene = new Scene(rootNode, 300, 200);
    // Określa scenę używaną przez obszar roboczy.
    myStage.setScene(myScene);
    // Wyświetla obszar roboczy oraz scenę.
    myStage.show();
  }
}
```

Kompilacja i uruchamianie programów JavaFX

- Programy JavaFX są kompilowane tak samo jak wszystkie inne programy pisane w Javie – można użyć programu `javac`.
- Niemniej jednak w zależności od docelowego środowiska wykonawczego może się pojawić konieczność wykonania określonych czynności dodatkowych – z tego względu wykorzystanie IDE, które potrafi w pełni obsługiwać platformę JavaFX jest najprostszym rozwiązaniem.
- Istnieje możliwość uruchamiania tego samego programu w wielu różnych środowiskach wykonawczych – można użyć programu `java`.
- Niemniej jednak w niektórych przypadkach mogą być potrzebne różne pliki pomocnicze, na przykład plik HTML lub plik JNLP (Java Network Launch Protocol).

Wątek aplikacji

- Nie można używać metody `init()` do tworzenia obszaru roboczego ani sceny. Obiektów tych nie można także tworzyć w konstruktorze aplikacji.
- Powodem tych ograniczeń jest konieczność tworzenia obu tych obiektów w wątku aplikacji.
- A jak się okazuje, zarówno metoda `init()`, jak i konstruktor aplikacji są wywoływane w wątku głównym, nazywanym także wątkiem uruchomieniowym. I właśnie z tego powodu ani konstruktora aplikacji, ani metody `init()` nie można używać do tworzenia obiektów `Stage` i `Scene`.

Wątek aplikacji

- Wszelkie zmiany w aktualnie wyświetlanym interfejsie użytkownika muszą być wykonywane w wątku aplikacji.
- Dodatkowo także wszystkie zdarzenia w aplikacjach JavaFX są generowane w wątku aplikacji. Dzięki temu procedur obsługi zdarzeń można używać do interakcji z graficznym interfejsem użytkownika aplikacji.
- Także metoda `stop()` jest wywoływana w wątku aplikacji.

Prosta kontrolka JavaFX: Label

- JavaFX udostępnia bogaty zestaw kontrolek. Najprostszą z nich jest etykieta, która pozwala wyświetlić komunikat lub obrazek.
- W JavaFX etykiety są instancjami klasy `Label` należącej do pakietu `javafx.scene.control`.
- Przykład:

```
// Tworzymy etykietę.
Label myLabel =
  new Label("JavaFX ma duże możliwości");
// Dodajemy etykietę do grafu sceny.
rootNode.getChildren().add(myLabel);
```

Stosowanie przycisków i zdarzeń

- Podstawową klasą związaną ze zdarzeniami JavaFX jest klasa `Event` zdefiniowana w pakiecie `javafx.event`.
- Klasa `Event` dziedziczy po `java.util.EventObject`, a to oznacza, że zdarzenia JavaFX mają te same podstawowe możliwości funkcjonalne co inne zdarzenia w języku Java.
- Aby obsłużyć zdarzenie, w pierwszej kolejności należy zarejestrować jego obiekt nasłuchujący, który będzie oczekiwać na zdarzenia. W momencie zgłoszenia zdarzenia zostaje wywołany odpowiedni obiekt nasłuchujący, który musi odpowiedzieć na nie i zakończyć działanie.

Stosowanie przycisków i zdarzeń

- Zdarzenia są obsługiwane poprzez zaimplementowanie interfejsu `EventHandler`, zadeklarowanego w pakiecie `javafx.event`:
  `interface EventHanderl<T extends Event>`
  gdzie `T` jest typem obsługiwanego zdarzenia.
- Interfejs ten deklaruje tylko jedną metodę `handle()`, której parametrem jest obiekt zdarzenia:
  `void handle(T eventObj)`
- W tym przypadku parametr `eventObj` jest zgłoszonym zdarzeniem.
- Zazwyczaj obiekty nasłuchujące zdarzeń są implementowane przy wykorzystaniu anonimowych klas wewnętrznych lub wyrażeń lambda.

Stosowanie przycisków i zdarzeń

- Przyciski są tworzone przy użyciu klasy `Button`, zdefiniowanej w pakiecie `javafx.scene.control`.
- Naciśnięcie przycisku spowoduje wygenerowanie zdarzenia `ActionEvent`. Klasa `ActionEvent` została zdefiniowana w pakiecie `javafx.event`. Obiekt nasłuchujący tego zdarzenia można zarejestrować, wywołując metodę `setOnAction()` obiektu `Button`.
- Metoda `setOnAction()` ustawia wartość właściwości `onAction` przechowującej referencję obiektu nasłuchującego.

Stosowanie przycisków i zdarzeń

```
// Tworzymy etykietę.
Label response = new Label("Przycisk");
// Tworzymy dwa przyciski.
Button btnUp = new Button("Góra");
Button btnDown = new Button("Dół");
// Obsługa zdarzenia ActionEvent przycisku Góra.
btnUp.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent ae) {
    response.setText("Nacisnąłeś przycisk Góra.");
  }
});
// Obsługa zdarzenia ActionEvent przycisku Dół.
btnDown.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent ae) {
    response.setText("Nacisnąłeś przycisk Dół.");
  }
});
// Dodajemy etykietę i przyciski do grafu sceny.
rootNode.getChildren().addAll(btnUp, btnDown, response);
```

Pola wyboru

- Pola wyboru są reprezentowane przez klasę `CheckBox`. Jej bezpośrednią klasą bazową jest `ButtonBase`.
- Pola wyboru reprezentowane przez klasę `CheckBox` mogą mieć trzy stany: pole zaznaczone, niezaznaczone i nieznany (nazywany także stanem niezdefiniowanym).
- Metoda `isSelected()` zwraca wartość `true`, jeśli pole wyboru jest zaznaczone, bądź wartość `false`, jeśli zaznaczenia nie ma.

Pola wyboru

```
Label response = new Label("");
CheckBox cbTablet = new CheckBox("Tablet");
// Obsługa zdarzeń ActionEvent pól wyboru.
cbTablet.setOnAction(new EventHandler<ActionEvent>() {
  public void handle(ActionEvent ae) {
    if(cbSmartphone.isSelected())
      response.setText("Zaznaczono pole 'smartfon'.");
    else
      response.setText("Usunięto 'smartfon'.");
    showAll();
  }
});
```

Pola wyboru

- Stan nieznany, którego można użyć, by zasygnalizować, że stan pola nie został jeszcze określony bądź też nie nadaje się on do użycia w bieżącej sytuacji, nie jest domyślnie stosowany. Aby włączyć możliwość stosowania stanu nieznanego w polu wyboru, należy wywołać metodę `setAllowIndeterminate()`.
- Aby określić, czy pole wyboru znajduje się w stanie nieznanym, należy wywołać metodę `isIndeterminate()`.

Listy

- Listy są reprezentowane przez obiekty klasy `ListView`.
- Kontrolki te wyświetlają listę elementów i pozwalają na zaznaczenie jednego bądź kilku z nich.
- Jedną z bardzo przydatnych cech kontrolek `ListView` jest automatyczne wyświetlanie pasków przewijania.
- ListView jest klasą sparametryzowaną `ListView<T>`, przy czym `T` określa typ elementów przechowywanych na liście.
- Domyślnie kontrolki `ListView` pozwalają na zaznaczanie tylko jednego elementu listy w danej chwili. Możliwość zaznaczania wielu elementów można włączyć.

Listy

- Kontrolki `ListView` można używać na dwa podstawowe sposoby:
  - Pierwszy z nich polega na zignorowaniu zdarzeń generowanych przez listę i pobieraniu jej zaznaczonych elementów wtedy, gdy program będzie ich potrzebować.
  - Drugi sposób bazuje na monitorowaniu zmian w zaznaczonych elementach listy poprzez zarejestrowanie odpowiednich obiektów nasłuchujących.
- Obiekt nasłuchujący zdarzeń generowanych przez listy korzysta z interfejsu `ChangeListener` zdefiniowanego w pakiecie `javafx.beans.value`. Interfejs ten definiuje tylko jedną metodę `change()`.

Listy

```
ObservableList<String> computerTypes =
  FXCollections.observableArrayList("Smartfon", "Tablet",
  "Notebook", "Stacjonarny" );
// Tworzymy kontrolkę ListView.
ListView<String> lvComputers = new ListView<String>(computerTypes);
// Określamy preferowaną wysokość i szerokość.
lvComputers.setPrefSize(100, 70);
// Pobieramy model zaznaczania elementów listy.
MultipleSelectionModel<String> lvSelModel =
  lvComputers.getSelectionModel();
// Tworzy obiekt nasłuchujący zdarzeń, który będzie odpowiadał na
// zmiany zaznaczenia elementów listy.
lvSelModel.selectedItemProperty().addListener(
  new ChangeListener<String>() {
    public void changed(ObservableValue<? extends String> changed,
      String oldVal, String newVal) {
        // Wyświetla nazwę wybranego elementu listy.
        response.setText("Wybranym typem komputera jest: " + newVal);
    }
});
```

Literatura (JavaBeans)

- H.Schildt: *Java. Przewodnik dla początkujących. Wydanie 7. Rozdział 18: Wprowadzenie do JavaFX.* Wydawnictwo HELION, Gliwice 2018.
- Java Platform, Client Technlogies: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm