Міністерство освіти і науки України
Волинський національний університет імені Лесі Українки
Навчально-науковий фізико-технологічний інститут

В. П. Муляр

ПРОЄКТУВАННЯ І РОЗРОБКА КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

Конспект лекцій

Луцьк
Вежа-друк
2022

УДК 004.438(07)
М 90

Рекомендовано до друку науково-методичною радою Волинського національного університету імені Лесі Українки (протокол № 3 від 16.11.2022 р.)

Рецензенти:

Яцюк С. М. – кандидат педагогічних наук, доцент кафедри загальної математики та методики навчання інформатики, декан факультету інформаційних технологій і математики Волинського національного університету імені Лесі Українки;

Багнюк Н. В. – кандидат технічних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету.

Муляр В. П.

М 90 Проєктування і розробка користувацьких інтерфейсів: конспект лекцій. Луцьк : Вежа-Друк, 2022. 52 с.

У конспекті лекцій розкрито теоретичні основи розробки користувацьких інтерфейсів на основі технології JavaFX. Розглянуто основні компоненти графічного інтерфейсу користувача. На конкретних прикладах розглянуто їх використання при створенні JavaFX-додатків. Належну увагу приділено анімації і трансформації зображень в JavaFX.

Для студентів ЗВО педагогічного та технічного спрямування, які вивчають дисципліни «Об'єктно-орієнтоване програмування», «Програмування на Java», «Алгоритми і структури даних», «Комп'ютерна фізика», «Моделювання фізичних явищ і процесів», а також для тих, хто цікавиться проєктуванням та розробкою користувацьких інтерфейсів.

УДК 004.438(07)

ЗМІСТ

Вступ

JavaFX є потужним інструментальним засобом для розробки графічного інтерфейсу користувача високої продуктивності, що включає аудіо, відео, графіку та анімацію. Для обробки подій використовуються лямбда-вирази, що істотно спрощує процес керування графічними інтерфейсами. Механізм прив'язки дозволяє реалізувати автоматичне оновлення однієї властивості при зміні іншої.

Технологія JavaFX надає можливість змінювати зовнішній вигляд інтерфейсу користувача за допомогою таблиць стилів CSS, що, як правило, зручніше, ніж надання атрибутів FXML-розмітки або виклик методів в Java.

Платформа JavaFX забезпечує створення двох видів анімації – анімацію по ключовим кадрам і анімацію з вбудованою часовою шкалою. Пакет javafx.scene.transform забезпечує трансформацію вузлів графа сцени, яка складається з афінних перетворень: обертання, переміщення, масштабування і зсуву. На відмінну від анімації, трансформація графічних об'єктів не має плавного видимого переходу від початкової до кінцевої точки під час певного проміжку часу, а виконується відразу. JavaFX містить удосконалені елементи управління, які забезпечують побудову діаграм, опрацювання мультимедійних даних тощо.

Конспект лекцій містить теоретичні та практичні основи розробки графічного інтерфейсу користувача з використанням технології JavaFX. В ньому розкрито структуру JavaFX-додатків, розглянуто використання лямбда-виразів для обробки подій. Розглянуто особливості зміни зовнішнього вигляду інтерфейсу користувача за допомогою таблиць стилів CSS. Належну увагу приділено анімації і трансформації зображень в JavaFX.

Навчальне видання буде корисним для студентів ЗВО педагогічного та технічного спрямування, які вивчають дисципліни «Об'єктно-орієнтоване програмування», «Програмування на Java», «Алгоритми і структури даних», «Комп'ютерна фізика», «Моделювання фізичних явищ і процесів», а також для тих, хто цікавиться проєктуванням та розробкою користувацьких інтерфейсів.

Лекція 1. Архітектура JavaFX. GUI-компонент Button

План

1. Модель програмування додатків платформи JavaFX
2. Програмний інтерфейс JavaFX API
3. Кнопка Button та її властивості.
4. Приклад створення JavaFX-додатку, що містить кнопки.

Модель програмування додатків платформи JavaFX 2.0

Один і той же код JavaFX-додатку може запускатися як настільний додаток, який розгортається на клієнтському комп'ютері автономно, може розгортатися як додаток *Java Web Start* або відображатися у Web-браузері як JavaFX-аплет, вбудований в HTML-сторінку.

Точкою входу в JavaFX-додаток служить Java-клас, що розширює абстрактний клас javafx.application.Application і метод main():

```
public class JavaFXApp extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void init(){
    //Ініціалізація додатку
    . . .
  }
  @Override
  public void start(Stage primaryStage) {
    //Установка параметрів сцени
    . . .
    primaryStage.setScene(scene);
    primaryStage.setVisible(true);
  }
  public void stop(){
    //Звільнення ресурсів додатку
    . . .
  }
}
```

У методі main() головного класу JavaFX-додатку викликає метод launch() класу Application, що відповідає за завантаження JavaFX-додатку. Крім того, головний клас JavaFX-додатку повинен перевизначити абстрактний метод start() класу Application, що

забезпечує створення та відображення сцени JavaFX-додатку.

Методи init() і stop() класу Application можуть використовуватись для ініціалізації даних та звільнення ресурсів JavaFX-додатку.

Оскільки метод init() викликається перед створенням головного потоку додатку JavaFX Application Thread, то ініціалізація JavaFX-додатку в методі init() за участю вузлів графа сцени має здійснюватися із застосуванням статичного методу javafx.application.Platform.runLater().

Для виконання JavaScript-коду на Web-сторінці, що містить JavaFX-додаток, головний клас JavaFX-додатку може використовувати метод getHostServices() класу Application та об’єкт netscape.javascript.JSObject.

Обробка вхідних аргументів або параметрів у головному класі JavaFX-додатку може бути здійснена за допомогою виклику методу getParameters() класу Application.

Поліпшити відображення та обробку процесу запуску та завантаження JavaFX-додатку можна декількома способами. Перший спосіб це використання обробника onGetSplash JavaScript-бібліотеки Deployment Toolkit API для створення заставки запуску JavaFX-аплету, вбудованого в Web-сторінку. Інший спосіб – це застосування CSS-стилів до Preloader-завантажувача за замовчуванням. І нарешті, можна створити свій клас передзавантажувача, що розширює абстрактний клас javafx.application.Preloader, і послатися на нього в JNLP-дескрипторі розгортання JavaFX-додатку. При цьому для зв’язку головного класу JavaFX-додатка із завантажувачем можна використовувати метод notifyPreloader() класу Application.

Метод start() класу Application містить як параметр об’єкт javafx.stage.Stage, що представляє графічний контейнер головного вікна JavaFX-додатку. Цей об’єкт Stage створюється середовищем виконання під час запуску JavaFX-додатку і передається в метод start() головного класу JavaFX-додатку, що дозволяє використовувати методи об’єкта Stage для встановлення та відображення сцени JavaFX-додатку. Замість об’єкта Stage, аргументу методу start(), розробник може створити свій екземпляр класу Stage для відображення сцени JavaFX-додатку.

Перед встановленням та відображенням сцени у графічному контейнері Stage головного вікна JavaFX-додатку необхідно створити граф сцени, що складається з кореневого вузла та його дочірніх елементів, та на його основі створити об’єкт javafx.scene.Scene сцени.

Як правило, як кореневий сайт використовується об’єкт

javafx.scene.Group, який створюється за допомогою конструктора і виступає як аргумент конструктора при створенні об'єкта javafx.scene.Scene.

Дочірні вузли графа сцени, що представляють графіку, елементи контролю GUI-інтерфейсу, медіаконтент, додаються до кореневого вузол за допомогою методу getChildren().add() або методу getChildren().addAll(). При цьому дочірні вузли можуть мати візуальні ефекти, режими накладання, CSS-стилі, прозорість, трансформації, обробники подій, брати участь в анімації за ключовими кадрами, анімації та ін.

Програмний інтерфейс JavaFX API

Програмний інтерфейс JavaFX API дає можливість розробляти Rich Client Application (RIA) додатки з насиченим графічним інтерфейсом, код яких поєднує широкі можливості платформи Java з багатою графікою та медіафункціональністю платформи JavaFX.

Компоненти графічного інтерфейсу користувача платформи JavaFX представлені такими пакетами JavaFX API, як javafx.scene.control, javafx.scene.chart, javafx.scene.image, javafx.scene.layout, javafx.scene.media, javafx.scene.shape, javafx.scene.text, javafx.scene.web та javafx.stage.

Усі компоненти GUI-інтерфейсу є об'єктами Node вузлів графа сцени та характеризуються ідентифікатором, CSS-стилем, межами, візуальними ефектами, прозорістю, трансформаціями, обробниками подій, станом, режимом накладання та участю в анімації.

Пакет javafx.scene.control надає такі GUI-компоненти, як панель Accordion, кнопку Button, прапорець CheckBox, список ChoiceBox, контекстне меню ContextMenu, гіперпосилання Hyperlink, мітку Label, список ListView, меню Menu, панель MenuBar, кнопку MenuButton, поле введення пароля PasswordField, індикатор ProgressBar, індикатор ProgressIndicator, перемикач RadioButton, панель ScrollPane, прокрутку ScrollBar, роздільник Separator, бігунок Slider, кнопку SplitMenuButton, панель SplitPane, таблицю TableView, панель із вкладками TabPane, багаторядкове поле TextArea, поле введення TextField, панель TitledPane, кнопку ToggleButton, групу ToggleGroup, панель ToolBar, вікно підказки Tooltip, дерево TreeView.

Пакет javafx.scene.chart забезпечує створення діаграм PieChart, AreaChart, BarChart, BubbleChart, LineChart та ScatterChart.

Пакет javafx.scene.image містить GUI-компонент зображення ImageView.

Пакет javafx.scene.layout надає панелі компонування AnchorPane, BorderPane, FlowPane, GridPane, HBox, StackPane, TilePane, VBox.

Пакет javafx.scene.media містить GUI-компоненти медіаконтенту MediaView та аудіоконтенту AudioClip.

Пакет javafx.scene.shape забезпечує малювання геометричних форм за допомогою таких GUI-компонентів, як Arc, Circle, CubicCurve, Ellipse, Line, Path, Polygon, Polyline, QuadCurve, Rectangle, SVGPath та Path.

Пакет javafx.scene.text містить GUI-компонент тексту Text.

Пакет javafx.scene.web забезпечує відображення HTML-контенту за допомогою GUI-компонента WebView та редагування HTML-контенту за допомогою GUI-компонента HTMLEditor.

Пакет javafx.scene містить групу Group та сцену Scene.

Пакет javafx.stage надає GUI-компоненти вікон Stage, Popup та FileChooser.

Кнопка Button та її властивості

Компонент Button представлений класом javafx.scene.control.Button, екземпляр якого може бути створений за допомогою класу-фабрики ButtonBuilder або за допомогою конструктора:

Button btn = New Button ();

Клас Button має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layout, layout, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– власні властивості cancelButton та defaultButton.

Приклад створення JavaFX-додатку з компонентом Button

1. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить кнопки *Button*, відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0 та в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationButton*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationButton*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationButton extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
```

```
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("Hello World!");
            primaryStage.setScene(scene);
            primaryStage.show();
        }
        public static void main(String[] args) {
            launch(args);
        }
    }
```

2. Змінюємо заголовок основного вікна JavaFX-додатку:

```
primaryStage.setTitle("Тестування GUI-компонентів");
```

3. Змінюємо кореневий вузол графа сцени та на його основі екземпляр сцени:

```
Group root = new Group ();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
```

4. Змінюємо властивості кнопки *Button*, яка поміщається в лівий верхній кут основного вікна JavaFX-додатку, встановлюємо текст кнопки:

```
Button btn;
btn = new Button();
btn.setLayoutX(20);
btn.setLayoutY(20);
btn.setText("Тестувати властивості");
```

5. Клацаємо ліворуч від обробника подій кнопки, вибираємо «використовувати лямда-вираз» та змінюємо його:

```
btn.setOnAction(e -> {
System.out.println("Властивості, успадковані від класу Node:");
System.out.println("Властивість blendMode:
"+btn.blendModeProperty().getValue());
 System.out.println("Властивість boundsInLocal:
"+btn.boundsInLocalProperty().getValue());
});
```

6. Створюємо другу кнопку, яка розміщується нижче за першу. Для другої кнопки встановлюємо текст, стиль, розміри:

```
Button btnON = new Button();
  btnON.setLayoutX(20);
  btnON.setLayoutY(150);
  btnON.setText("Встановити властивості");
  btnON.setStyle("-fx-font: bold italic 12pt Arial;-fx-text-fill:
```

```
#660000;"
        + "-fx-background-color: #ff99ff; -fx-border-width: 3px; "
        + "-fx-border-radius: 30;-fx-background-radius: 30;"
        + "-fx-border-color: # 660066; ");
    btnON.setPrefSize(250,30);
```

7. Створюємо обробник подій для другої кнопки, який встановлює такі властивості першої кнопки, як режим накладання, маска, курсор миші, візуальний ефект, управління компонуванням, прозорість, обертання, переміщення, масштабування, розміри, підказка, значок, стиль, вирівнювання, підкреслення тексту, перенесення рядків, прив’язка до клавіші активації, задній план:

```
    btnON.setOnAction(e->{
      btn.setBlendMode(BlendMode.DARKEN);
      javafx.scene.shape.Circle clip = new
javafx.scene.shape.Circle(75,53,80);
      //btn.setClip(clip);
      btn.setCursor(Cursor.CLOSED_HAND);
      DropShadow effect=new DropShadow();
      effect.setOffsetX(10);
      effect.setOffsetY(10);
      btn.setEffect(effect);
      //btn.setManaged(false);
      //btn.setMouseTransparent(true);
      btn.setOpacity(0.5);
      btn.setRotate(10);
      btn.setLayoutX(80);
      btn.setScaleX(1.8);
      btn.setLayoutY(170);
      btn.setTranslateZ(-50);
      btn.setPrefSize(150,100);
      btn.setTooltip(new Tooltip("Це кнопка тестування властивостей
класу Button"));
      Image im = new
Image(this.getClass().getResource("image.png").toString());
      ImageView imv = new ImageView(im);
      imv.setFitHeight(50);
      imv.setFitWidth(50);
      btn.setGraphic(imv);
      btn.setStyle("-fx-font: bold italic 8pt Helvetica;");
```

```
        //btn.setFont(Font.font("Helvetica", FontWeight.BOLD,
        // FontPosture.ITALIC, 10));
        btn.setAlignment(Pos.CENTER);
        btn.setContentDisplay(ContentDisplay.RIGHT);
        btn.setUnderline(true);
        btn.setWrapText(true);
        //btn.setCancelButton(true);
        //btn.toBack();
    });
```

8. Скачуємо з інтернету рисунок із зображенням смайлика, зберігаємо його як *image.png* у папці, де розміщений файл JavaFXApplicationButton.java.

9. Додаємо другу кнопку в кореневий вузол:

```
root.getChildren().add(btnON);
```

10. Запускаємо створений JavaFX-додаток, клацнувши на піктограмі *Виконати*.

В результаті можна побачити дві кнопки (рис. 1), натискання однієї з яких дозволить вивести в консоль значення властивостей цієї кнопки, а натискання іншої кнопки призведе до зміни першої кнопки (рис. 2).

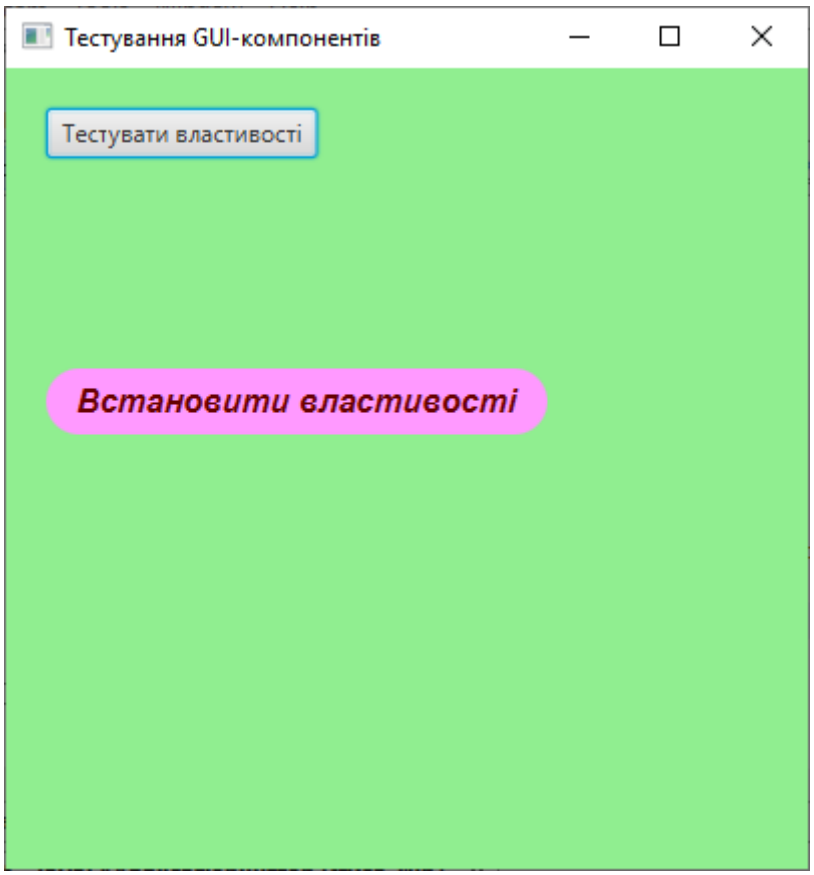

Рис. 1. JavaFX-додаток з GUI-інтерфейсом, що містить дві кнопки

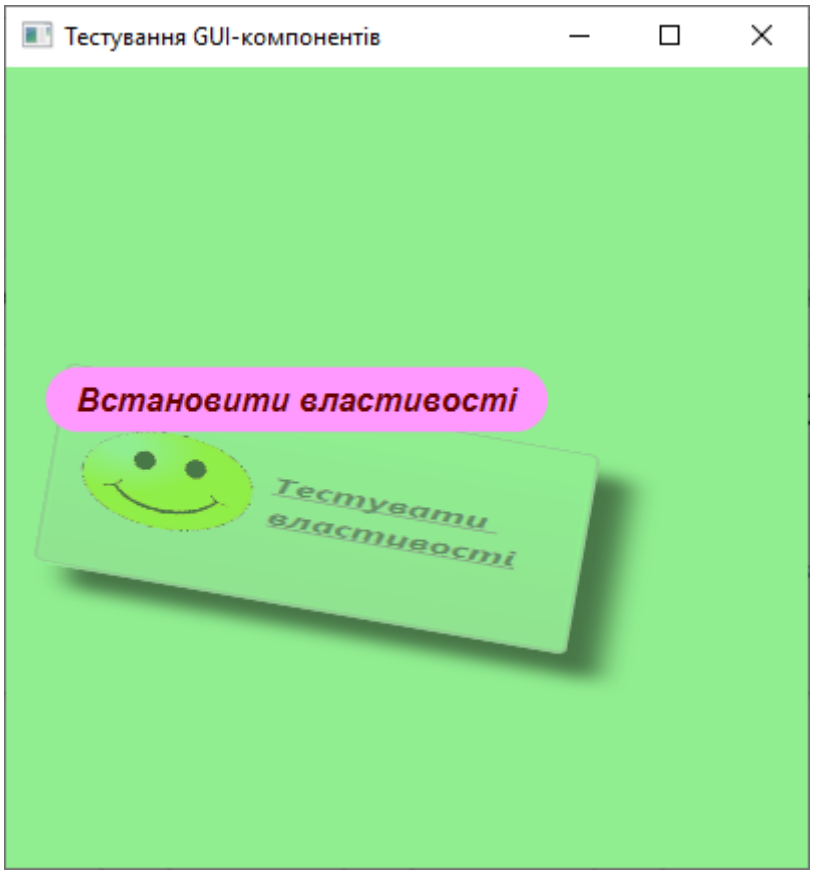

Рис. 2. Зміна властивостей першої кнопки за допомогою натискання другої кнопки

Натиснувши кнопку *Тестувати властивості* до її зміни та після зміни кнопки, у вікні *Виведення* середовища NetBeans можна побачити початкові та змінені значення властивостей кнопки.

Контрольні запитання

1. Яка модель програмування додатків на платформі JavaFX?
2. Яке призначення інтерфейсу JavaFX API?
3. Яке призначення компоненту *Button*?
4. Які основні властивості притаманні компоненту *Button*?

Лекція 2. GUI-компонент CheckBox

План

1. Компонент CheckBox та його властивості.
2. Приклад створення JavaFX-додатку з компонентом CheckBox.

Компонент CheckBox та його властивості

Компонент *CheckBox* представлений класом javafx.scene.control.CheckBox, екземпляр якого може бути створений за допомогою класу-фабрики *CheckBoxBuilder* або за допомогою

одного з конструкторів:

CheckBox ckb = new CheckBox();

CheckBox ckb = new CheckBox("[текст]");

Клас CheckBox має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– власні властивості indeterminate, selected і allowIndeterminate.

Приклад створення JavaFX-додатку з компонентом CheckBox

1. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить прапорець *CheckBox*, відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0 та в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationCheckBox*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationCheckBox*, який розширює клас Application.

package javafxapplicationbutton;

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationCheckBox extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start* вилучіть весь код.

3. Установіть заголовок основного вікна JavaFX-додатку:

```
primaryStage.setTitle("Тестування GUI-компонентів: CheckBox");
```

4. Установіть кореневий вузол графа сцени та на його основі екземпляр сцени:

```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
```

5. Створіть екземпляр прапорця CheckBox із текстом. Прапорець розміщується в лівому верхньому куті основного вікна JavaFX-додатку. Для нього визначаємо обробник подій натискання кнопки

миші таким чином, що при натисканні кнопкою миші на прапорці в консоль виводяться значення властивостей CheckBox:

```
CheckBox ckb = new CheckBox("Тестувати властивості");
// ckb=CheckBoxBuilder.create().build();
ckb.setLayoutX(20);
ckb.setLayoutY(20);
ckb.setOnMousePressed(e->{
System.out.println("Властивості, успадковані від класу Node:");
  System.out.println("Властивість blendMode: "+
ckb.blendModeProperty().getValue());
  System.out.println("Властивість boundsInLocal: "+
ckb.boundsInLocalProperty().getValue());
});
```

6. Створіть кнопку, яка розміщується нижче за прапорець, при натисканні якої змінюються властивості прапорця CheckBox. Для кнопки встановлюється текст, стиль, переважні розміри:

```
Button btnON = new Button();
btnON.setLayoutX(20);
    btnON.setLayoutY(100);
btnON.setText("Установити властивості");
btnON.setStyle("-fx-font: bold italic 12pt Arial;"
   + "-fx-text-fill: white;-fxbackground-color: #0000cc;"
   + "-fx-border-width: 3px; "
   + "-fx-border-color:#6699ff #000066 #000066 #6699ff;" );
btnON.setPrefSize(200,30);
```

7. Для кнопки встановлюємо обробник подій. В ньому, якщо встановлено прапорець CheckBox, визначаються такі властивості CheckBox, як режим накладання, маска, курсор миші, візуальний ефект, управління компонуванням, активація мишею, прозорість, переміщення, масштабування, переважні розміри, підказка, значок, стиль, вирівнювання, підкреслення тексту, перенесення рядків, встановлення трьох станів прапорця, задній план та поворот:

```
btnON.setOnAction(e->{
  if(ckb.selectedProperty().getValue()==true){
    ckb.setBlendMode(BlendMode.HARD_LIGHT);
    Rectangle clip = new Rectangle(0,15,15,20);
    //ckb.setClip(clip);
    ckb.setCursor(Cursor.CROSSHAIR);
    DropShadow effect=new DropShadow();
```

```
            effect.setOffsetX(5);
            effect.setOffsetY(10);
            ckb.setEffect(effect);
            //ckb.setManaged(false);
            //ckb.setMouseTransparent(true);
            //ckb.setOpacity(0.5);
            ckb.setLayoutX(50);
            ckb.setTranslateZ(-50);
            ckb.setScaleX(1.8);
            ckb.setPrefSize(150,50);
            ckb.setTooltip(new Tooltip ("Це перемикач тестування
властивостей класу CheckBox"));
            Image im = new
Image(this.getClass().getResource("image.png").toString());
            ImageView imv = new ImageView(im);
            imv.setFitHeight(50);
            imv.setFitWidth(50);
            ckb.setGraphic(imv);
            ckb.setStyle("-fx-font: bold italic 10pt Helvetica;");
            ckb.setAlignment(Pos.CENTER);
            ckb.setContentDisplay(ContentDisplay.RIGHT);
            ckb.setUnderline(true);
            ckb.setWrapText(true);
            ckb.setAllowIndeterminate(true);
            //ckb.toBack();
            //ckb.setTranslateY(50);
          }
        });
```

8. Скачайте з інтернету рисунок із зображенням смайлика та збережіть його як *image.png* у папці, де розміщений файл JavaFXApplicationCheckBox.java.

9. Прапорець і кнопку додайте в кореневий вузол графа сцени, для об'єкта Stage встановіть створену сцену, і об'єкт Stage встановіть видимим:

```
root.getChildren().add(btnON);
root.getChildren().add(ckb);
primaryStage.setScene(scene);
primaryStage.show();
```

10. Запустіть створений JavaFX-додаток, клацнувши правою

кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна побачити прапорець та кнопку (рис. 3). При клацанні мишею на прапорці буде змінено його стан і в консоль буде виведено значення властивостей прапорця, а при натисканні кнопки стан прапорця зміниться (рис. 4).

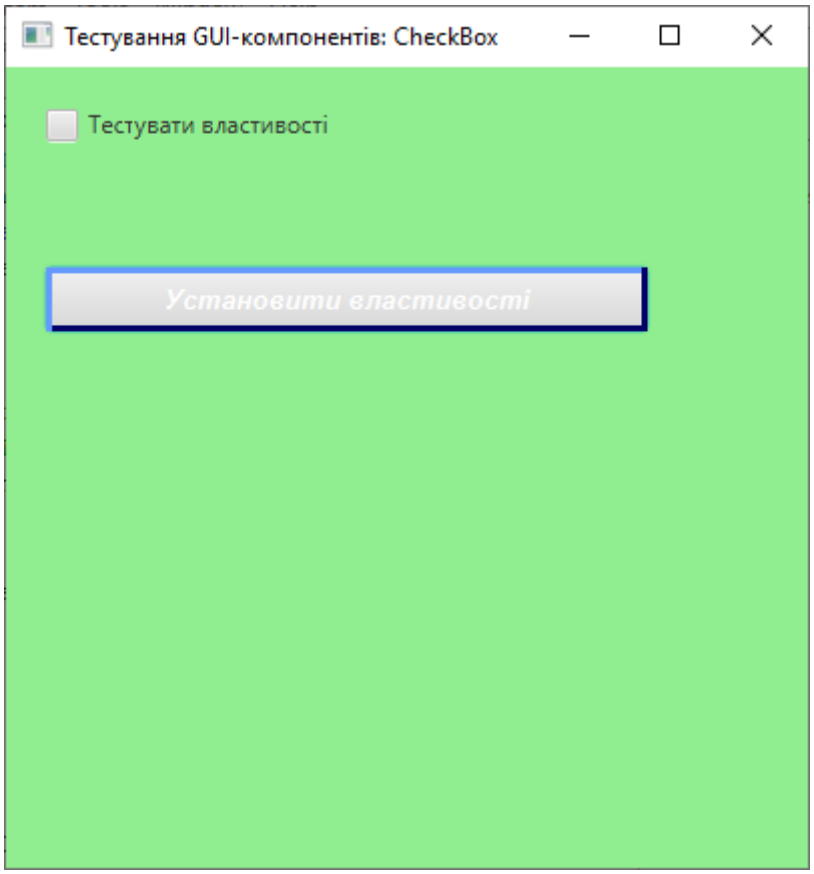

Рис. 3. JavaFX-прикладання з GUI-інтерфейсом, що містить прапорець і кнопку

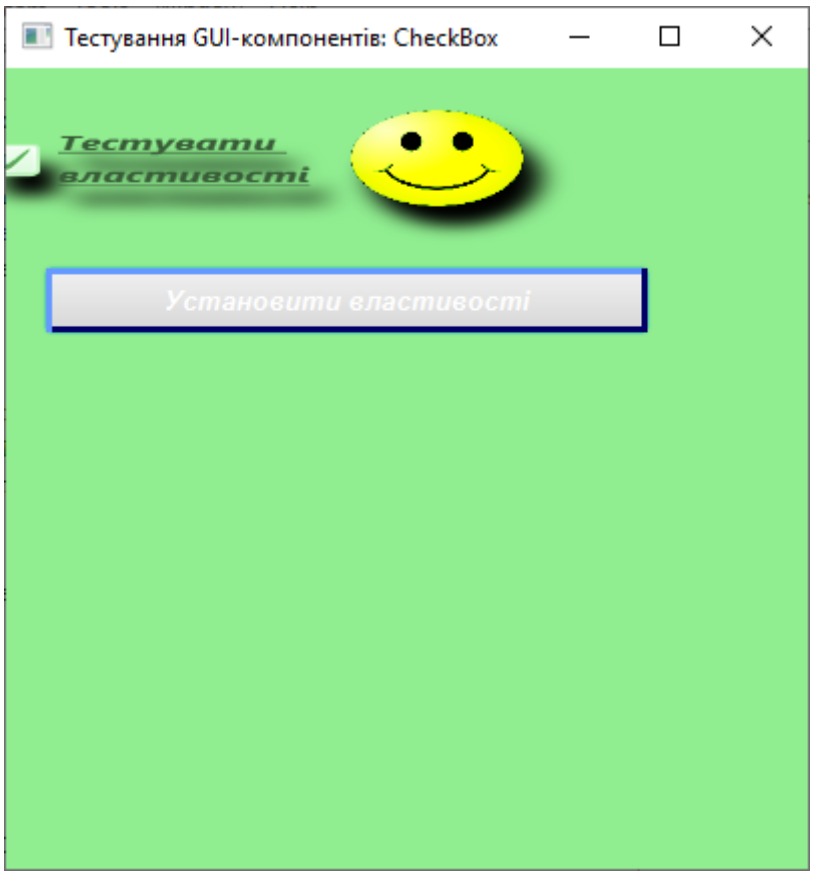

Рис. 4. Зміна властивостей прапорця за допомогою натискання кнопки

Властивості компонента *CheckBox*, успадковані від класів *ButtonBase*, *Labeled*, *Control*, *Parent* і *Node*, аналогічні тим самим властивостям, які має компонент *Button*.

Окрім методу setOnAction() для визначення обробника подій компонента можна використовувати методи setOnMousePressed(), setOnMouseClicked() та setOnMouseReleased().

При використанні методу setOnAction() властивість armed набуває значення true, а властивість pressed – значення false. При використанні методу setOnMousePressed() властивості armed та pressed приймають значення true, а при застосуванні методів setOnMouseClicked() і setOnMouseReleased() ці властивості набувають значення false.

Крім того, обробку подій зміни стану перемикача можна здійснити, використовуючи JavaFX Beans-властивість компонента CheckBox:

```
ckb.selectedProperty().addListener(new ChangeListener<Boolean>()
{
    public void changed(ObservableValue<? extends Boolean> ov,
```

Boolean old_val, Boolean new_val) {

. . .

} });

Відмінність набору властивостей компонента *CheckBox* від набору властивостей компонента *Button* полягає в наявності властивостей самого класу javafx.scene.control.CheckBox *indeterminate*, *selected* і *allowIndeterminate*.

Властивість selected приймає значення *true* чи *false* залежно від того, чи встановлено прапорець чи ні.

Якщо властивість *allowIndeterminate* встановити рівним true, тоді замість галочки можна поставити межу біля прапорця. У цьому властивість *indeterminate* прийме значення *true*, а властивість *selected* – значення *false*.

Контрольні запитання

1. Яке призначення компоненту *CheckBox*?
2. Як створити екземпляр компоненту *CheckBox*?
3. Які основні властивості має компонент *CheckBox?*

Лекція 3. GUI-компонент RadioButton

План

1. Компонент RadioButton та його властивості.
2. Приклад створення JavaFX-додатку з компонентом RadioButton.

Компонент RadioButton та його властивості

Компонент RadioButton представлений класом javafx.scene.control.RadioButton, екземпляр якого може бути створений за допомогою класу-фабрики RadioButtonBuilder або за допомогою одного з конструкторів:

RadioButton btn = New RadioButton();

RadioButton btn = new RadioButton("[текст]");

Клас RadioButton має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased,

onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– успадковані від класу javafx.scene.control.ToggleButton властивості selected та toggleGroup.

Властивості компонента RadioButton аналогічні властивостям компонента ToggleButton.

Об'єднання перемикачів RadioButton у групу ToggleGroup відрізняється від об'єднання кнопок ToggleButton у групу ToggleGroup.

У групі перемикачів RadioButton щонайменше один елемент повинен бути у вибраному стані, і вибір іншого перемикача скасовує вибір решти. Таким чином, у групі перемикачів RadioButton постійно вибрано лише один елемент, і його не можна перевести у невибраний стан.

У групі кнопок ToggleButton при натисканні однієї з кнопок решта всіх кнопок групи автоматично віджимається, однак натиснуту кнопку можна віджати.

Екземпляр класу ToggleGroup можна створити за допомогою класу-фабрики ToggleGroupBuilder або за допомогою конструктора:

ToggleGroup tgroup = New ToggleGroup();

Клас ToggleGroup має властивість selectedToggle, що вказує на обраний в даний момент елемент групи.

Приклад створення JavaFX-додатку з компонентом RadioButton

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить перемикач *RadioButton*, в меню *Файл* оберіть *Створити*

проект | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationRadioButton*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationRadioButton*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationRadioButton extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть

заголовок основного вікна JavaFX-додатку “Тестування GUI-компонентів: RadioButton” та зробіть видимим об’єкт Stage:

```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування GUI-компонентів:
RadioButton");
primaryStage.show();
```

4. Створіть групу ToggleGroup та перший екземпляр кнопки RadioButton з текстом. Кнопку помістіть у лівий верхній кут основного вікна JavaFX-програми. Для неї визначте такі властивості, як режим накладання, курсор миші, переважні розміри, спливаюча підказка, значок, стиль, вирівнювання, перенесення рядків, групу, фокус та обробник зміни значення selected кнопки, в якому змінюється загальний колір сцени:

```
ToggleGroup tgroup=new ToggleGroup();
RadioButton btnOn;
btnOn = new RadioButton("Mozilla Firefox");
btnOn.setLayoutX(20);
btnOn.setLayoutY(20);
btnOn.setBlendMode(BlendMode.MULTIPLY);
btnOn.setCursor(Cursor.CLOSED_HAND);
btnOn.setPrefSize(200,80);
btnOn.setTooltip(new Tooltip("Це кнопка вибору Mozilla
Firefox"));
Image imOn = new
Image(this.getClass().getResource("imageOn.jpg").toString());
ImageView imvOn = new ImageView(imOn);
imvOn.setFitHeight(50);
imvOn.setFitWidth(50);
btnOn.setGraphic(imvOn);
btnOn.setStyle("-fx-font: bold italic 12pt Georgia;");
btnOn.setAlignment(Pos.CENTER);
btnOn.setContentDisplay(ContentDisplay.RIGHT);
btnOn.setTextAlignment(TextAlignment.CENTER);
btnOn.setGraphicTextGap(10);
btnOn.setWrapText(true);
btnOn.setToggleGroup(tgroup);
btnOn.requestFocus();
```

```
btnOn.selectedProperty().addListener((javafx.beans.value.Observabl
eValue<? extends Boolean> ov, Boolean old_val, Boolean new_val) -> {
        if (new_val.equals(Boolean.TRUE))
            scene.setFill(Color.web("#fff8dc"));
    });
```

5. Створіть другий екземпляр кнопки RadioButton із текстом. Другу кнопку помістіть під першою кнопкою, і для неї визначте також такі властивості, як режим накладання, курсор миші, переважні розміри, підказка, значок, стиль, вирівнювання, перенесення рядків, група та обробник зміни значення властивості selected кнопки, в якому змінюється загальний колір сцени:

```
    RadioButton btnOf;
    btnOf = new RadioButton("Internet Explorer");
    btnOf.setLayoutX(20);
    btnOf.setLayoutY(100);
    btnOf.setBlendMode(BlendMode.MULTIPLY);
    btnOf.setCursor(Cursor.CLOSED_HAND);
    btnOf.setPrefSize(200,80);
    btnOf.setTooltip(new Tooltip("Це кнопка вибору Internet
Explorer"));
    Image imOf = new
Image(this.getClass().getResource("imageOf.jpg").toString());
    ImageView imvOf = new ImageView(imOf);
    imvOf.setFitHeight(50);
    imvOf.setFitWidth(50);
    btnOf.setGraphic(imvOf);
    btnOf.setStyle("-fx-font: bold italic 12pt Georgia;" );
    btnOf.setAlignment(Pos.CENTER);
    btnOf.setContentDisplay(ContentDisplay.RIGHT);
    btnOf.setTextAlignment(TextAlignment.CENTER);
    btnOf.setGraphicTextGap(10);
    btnOf.setWrapText(true);
    btnOf.setToggleGroup(tgroup);
    btnOf.selectedProperty().addListener((javafx.beans.value.Observable
Value<? extends Boolean> ov, Boolean old_val, Boolean new_val) -> {
        if (new_val.equals(Boolean.TRUE))
            scene.setFill(Color.web("#99ffff"));
    });
```

6. Скачайте з інтернету рисунки, назвавши як *imageOn.jpg* та

imageOf.jpg і збережіть їх у папці, де розміщений файл JavaFXApplicationRadioButton.java.

7. Створені кнопки додайте в кореневий вузол графа сцени:

root.getChildren().add(btnOn);

root.getChildren().add(btnOf);

8. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити дві кнопки RadioButton (рис. 5), при виборі яких змінюється колір сцени. Спочатку при відкритті вікна програми жодна з кнопок не вибрана, однак фокус наведено на першу кнопку. Після вибору однієї з кнопок вже не можна перевести сцену у вихідний стан із вихідним кольором та двома невибраними кнопками.

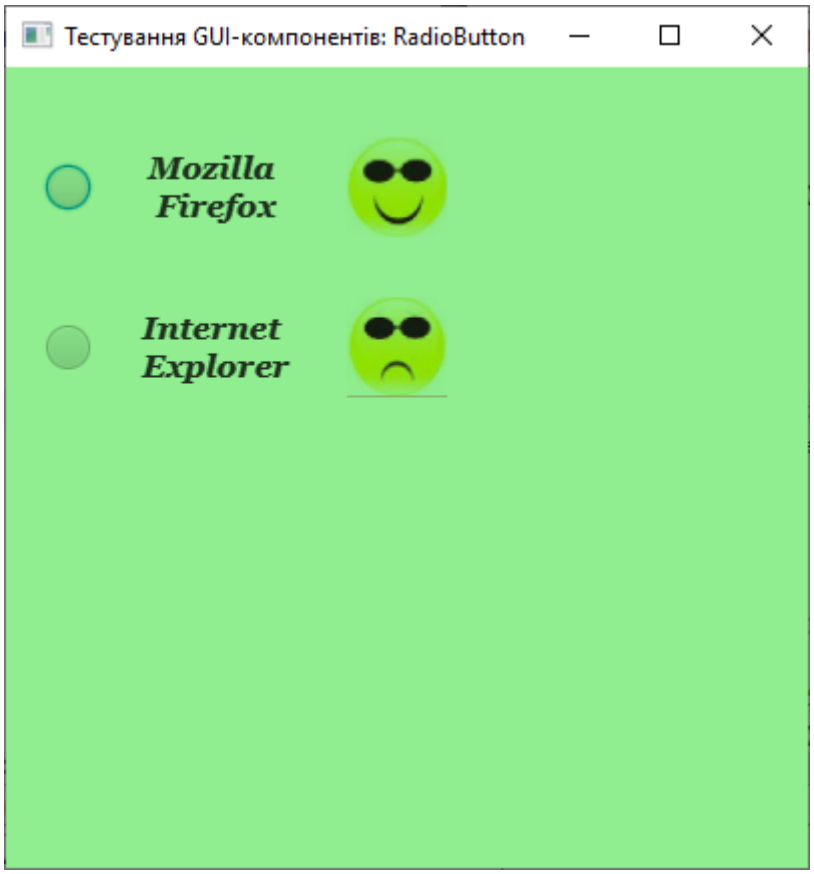

Рис. 5. JavaFX-додаток з GUI-інтерфейсом, що містить кнопки RadioButton

Контрольні запитання

1. Яке призначення компонента *RadioButton*?
2. Як створити екземпляр компонента *RadioButton*?

3. Які основні властивості має компонент *RadioButton?*

Лекція 4. Візуальні ефекти Blend, Bloom, Glow

План

1. Візуальні ефекти в JavaFX
2. Ефект змішування Blend.
3. Ефект світіння Bloom.
4. Ефект світіння Glow.
5. Приклад створення JavaFX-додатків, які містять візуальні ефекти змішування Blend, свічення Bloom та Glow.

Візуальні ефекти в JavaFX

Візуальні ефекти платформи JavaFX 2.0 представлені javafx.scene.effect. При цьому базовим класом всіх ефектів пакета javafx.scene.effect є абстрактний клас Effect, що має реалізації у вигляді класів Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DisplacementMap, DropShadow, GaussianBlur, Glow, ImageInput, InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone і Shadow.

Ефект, представлений Effect-об’єктом, пов’язується з вузлом графа сцени за допомогою методу setEffect() класу javafx.scene.Node та забезпечує створення нового зображення Node-вузла, отриманого в результаті модифікації вихідного графічного Node-вузла.

Ефект змішування Blend

Візуальний ефект змішування представлений у технології JavaFX 2.0 класом javafx.scene.effect.Blend.

Примірник класу Blend може бути створений за допомогою класу-фабрики javafx.scene.effect.BlendBuilder або за допомогою конструктора:

Blend blend = new Blend ();

Blend-ефект бере як один вхід зображення вузла Node, до якого ефект приєднаний за допомогою методу setEffect() класу Node, і змішує його з іншим ефектом Effect, який виступає як інший вхід Blend-ефекту. У цьому режим змішування визначається властивістю mode класу Blend.

Другий вхід Blend-ефекту, що містить об’єкт Effect, може бути двох типів – це може бути нижній або верхній вхід операції змішування. Effect-об’єкт встановлюється як нижній вхід за допомогою методу setBottomInput() класу Blend або як верхній вхід за

допомогою методу setTopInput() класу Blend.

Режим змішування зображення вузла Node з Effect-об’єктом встановлюється методом setMode() класу Blend, що приймає як аргумент поле SRC_OVER, SRC_IN, SRC_OUT, SRC_ATOP, ADD, MULTIPLY, SCREEN, OVERLAY, DARKEN, LIGHTEN, COLOR_DODGE, COLOR_BURN, HARD_LIGHT, SOFT_LIGHT, DIFFERENCE, EXCLUSION, RED, GREEN і BLUE перерахування javafx.scene.effect.BlendMode.

Крім властивостей mode, bottomInput та topInput клас Blend має властивість opacity, що визначає прозорість верхнього введення перед змішуванням, значення якого встановлюється за допомогою методу setOpacity() класу Blend.

Ефект світіння Bloom

Візуальний ефект світіння Bloom представлений у технології JavaFX 2.0 класом javafx.scene.effect.Bloom.

Примірник класу Bloom може бути створений за допомогою фабрики javafx.scene.effect.BloomBuilder або за допомогою конструктора:

Bloom bloom = new bloom();

Bloom-ефект бере як вхід зображення вузла Node, до якого ефект приєднаний за допомогою методу setEffect() класу Node, і засвічує яскраві ділянки графічного вмісту вузла Node, ґрунтуючись на значенні властивості threshold.

Властивість threshold класу Bloom визначає поріг яскравості пікселів, після якого вони будуть світитися, і може набувати значення від 0.0 до 1.0 (за замовчуванням 0.3).

Інша властивість input класу Bloom, значення якого встановлюється за допомогою методу setInput(), може визначати як вхід інший ефект Effect, створюючи, таким чином, ланцюжок ефектів.

Ефект світіння Glow

Візуальний ефект світіння Glow представлений у технології JavaFX 2.0 класом javafx.scene.effect.Glow.

Примірник класу Glow може бути створений за допомогою фабрики javafx.scene.effect.GlowBuilder або за допомогою конструктора:

Glow glow = new Glow();

Glow-ефект бере як вхід зображення вузла Node, до якого ефект

приєднаний за допомогою методу setEffect() класу Node, і засвічує графічний вміст вузла Node, ґрунтуючись на значенні властивості level.

Властивість рівня класу Glow визначає інтенсивність світіння і може набувати значення від 0.0 до 1.0 (за замовчуванням 0.3).

Інша властивість input класу Glow, значення якого встановлюється за допомогою методу setInput(), може визначати як вхід інший ефект Effect, створюючи, таким чином, ланцюжок ефектів.

Дія Glow-ефекту подібна до дії Bloom-ефекту, відрізняючись тим, що Glow-ефект засвічує зображення більш рівномірно.

Приклад створення JavaFX-додатків, які містять візуальні ефекти змішування Blend, свічення Bloom та Glow

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить візуальний ефект *Blend*, в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationBlendBloomGlow*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationBlendBloomGlow*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationBlendBloomGlow extends
Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
```

```
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start()* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування візуальних ефектів: Blend, Bloom, Glow” та зробіть видимим об’єкт Stage:

```
FlowPane root = new FlowPane();
root.setOrientation(Orientation.VERTICAL);
root.setAlignment(Pos.CENTER);
root.setVgap(8);
Scene scene = new Scene(root, 500, 300, Color.GRAY);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування візуальних ефектів: Blend, Bloom, Glow ");
primaryStage.show();
```

4. Створіть кнопку Button. При проходженні курсора миші через неї графічний вміст кнопки Button має поєднуватись із лінійним градієнтом за допомогою Blend-ефекту. В якості нижнього входу Blend-ефекту використайте ефект ColorInput, що забезпечує як вхід для іншого ефекту прямокутник, заповнений певним кольором:

```
final Button btn = new Button();
btn.setLayoutX(100);
btn.setLayoutY(100);
btn.setText("Blend");
btn.setPrefSize(100,50);
btn.setStyle("-fx-font: italic bold 14pt Georgia; "
```

```
                + "-fx-text-fill: white; -fxbackground-color: black;");
        Stop[] stops = new Stop[] {
            new Stop(0, Color.RED), new Stop(1, Color.YELLOW)
        };
        LinearGradient lg = new LinearGradient(0, 0, 0.25, 0.25, true,
CycleMethod.REFLECT, stops);
        ColorInput colorInput = new ColorInput();
        colorInput.setWidth(100);
        colorInput.setHeight(50);
        colorInput.setPaint(lg);
        final Blend blend = new Blend();
        blend.setMode(BlendMode.LIGHTEN);
        blend.setBottomInput(colorInput);
        btn.setOnMouseEntered((MouseEvent event) -> {
            btn.setEffect(blend);
        });
        btn.setOnMouseExited((MouseEvent event) -> {
            btn.setEffect(null);
        });
```

5. Створіть кнопку Button, через яку при проходженні курсора миші яскраві ділянки графічного вмісту кнопки Button засвічуються, створюючи ілюзію освітлення яскравим джерелом світла:

```
        final Button btn1 = new Button();
        btn1.setLayoutX(100);
        btn1.setLayoutY(100);
        btn1.setText("Bloom");
        btn1.setPrefSize(100,30);
        btn1.setStyle("-fx-font: bold italic 14pt Georgia; "
            + "-fx-text-fill: white; -fx-background-color: #a0522d;"
            + "-fx-border-width: 3px; "
            + "-fx-border-color: #f4a460#800000 #800000 #f4a460;");
        final Bloom bloom = new Bloom();
        bloom.setThreshold(0.3);
        btn1.setOnMouseEntered((MouseEvent event) -> {
            btn1.setEffect(bloom);
        });
        btn1.setOnMouseExited((MouseEvent event) -> {
            btn1.setEffect(null);
        });
```

6. Створіть кнопку Button, що має ореол світіння, що створюється за допомогою світіння прямокутника, розташованого під кнопкою. При цьому інтенсивність світіння має змінюватися з часом із використанням анімації:

```
Button btn2 = new Button();
btn2.setText("Glow");
btn2.setPrefSize(100,30);
btn2.setStyle("-fx-font: bold italic 14pt Georgia;"
    + "-fx-text-fill: white; "
    + "-fx-background-color: #a0522d;"
    + "-fx-border-width: 3px; "
    + "-fx-border-color:#f4a460 #800000 #800000 #f4a460;");
Glow glow = new Glow();
glow.setLevel(0.0);
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
KeyValue kv = new KeyValue(glow.levelProperty(), 1.0);
KeyFrame kf = new KeyFrame(Duration.millis(2000), kv);
timeline.getKeyFrames().add(kf);
timeline.play();
Rectangle r = new Rectangle();
r.setWidth(115);
r.setHeight(45);
r.setArcHeight(20);
r.setArcWidth(20);
r.setFill(Color.web("#f4a460"));
r.setStroke(Color.WHITE);
r.setStrokeWidth(5);
r.setStrokeType(StrokeType.OUTSIDE);
r.setEffect(glow);
```

4. Створіть екземпляр компонування StackPane із відповідними властивостями, додайте до нього об’єкти r та btn2.

```
StackPane pane = new StackPane();
pane.getChildren().addAll(r,btn2);
pane.setLayoutX(100);
pane.setLayoutY(100);
```

5. Створені об’єкти btn, btn1 та pane додайте в кореневий вузол графа сцени:

root.getChildren().addAll(btn, btn1, pane);

6. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити кнопки із візуальними ефектами Blend, Bloom і Glow (рис. 1).

Рис. 1. JavaFX-додаток, що містить кнопки з візуальними ефектами Blend, Bloom і Glow

Контрольні запитання

1. Яке призначення візуальних ефектів *Blend*, *Bloom* і *Glow*?
2. Як створити екземпляри ефектів *Blend*, *Bloom* і *Glow*?
3. Які основні властивості мають візуальні ефекти *Blend*, *Bloom* і *Glow*?

Лекція 5. Трансформація і анімація

План

1. JavaFX-анімація.
2. JavaFX-трансформація.
3. Приклад створення JavaFX-додатку, які містить трансформацію та анімацію.

JavaFX-анімація

Платформа *JavaFX* 2.0 забезпечує створення двох видів анімації – анімацію по ключовим кадрам і анімацію з вбудованою тимчасової

шкалою.

JavaFX-анімацію представляє пакет javafx.animation, базовим класом якого є клас *Animation*. Клас *Animation* розширюється класами *Timeline* і *Transition*, при цьому клас *Timeline* представляє анімацію по ключовим кадрам, а клас *Transition* – анімацію з вбудованою тимчасовою шкалою.

Клас *Animation* має набір властивостей, що дозволяють управляти швидкістю і напрямком анімації, затримкою і кількістю циклів анімації, встановлювати автореверс анімації, зчитувати статус анімації, обробляти завершення анімації та ін.

Швидкість і напрямок анімації можна встановити за допомогою методу *setRate(double value)*, затримку анімації – за допомогою методу *setDelay(Duration value)*, кількість циклів анімації – методом *setCycleCount(int value),* автореверс анімації – методом *setAutoReverse(boolean value)*, зчитати статус *Animation.Status.PAUSED, Animation.Status.RUNNING* або *Animation.Status.STOPPED* анімації – методом *getStatus()*, встановити обробник завершення анімації – методом *setOnFinished(EventHandler <ActionEvent> value).*

Також клас *Animation* надає методи управління життєвим циклом анімації:

jumpTo(Duration time) – перехід анімації до зазначеної позиції на часовій шкалі;

playFrom(Duration time) – запуск анімації, починаючи з вказаної позиції на часовій шкалі;

play() – запуск анімації з поточної позиції на часовій шкалі;

playFromStart() – запуск анімації з початкової позиції на часовій шкалі;

stop() – зупинка анімації;

pause() – пауза анімації.

Анімація по ключовим кадрам дозволяє створити видиму зміну значення будь-якої *JavaFX*-властивості за певний проміжок часу за допомогою класу *Timeline*.

Примірник класу *Timeline* можна створити за допомогою класу-фабрики *TimelineBuilder* або за допомогою одного з конструкторів, що дозволяють встановити частоту кадрів і набір ключових кадрів анімації:

public Timeline(double targetFramerate)

public Timeline(double targetFramerate, KeyFrame ... keyFrames)

public Timeline(KeyFrame ... keyFrames) і public Timeline()

Набір ключових кадрів *Timeline*-анімації можна поповнити методом *getKeyFrames().AddAll()*, а зупинити *Timeline*-анімацію і повернути її до початкової позиції – методом *stop()*.

Ключовий кадр *Timeline*-анімації представлений класом *javafx.animation.KeyFrame* і визначає зміни значень *JavaFX*-властивостей за певний проміжок часу.

Примірник класу *KeyFrame* можна створити за допомогою набору конструкторів, що дозволяють встановити час відтворення ключового кадру, ім'я ключового кадру, обробник закінчення ключового кадру і набір змін значень *JavaFX*-властивостей:

public KeyFrame(Duration time, java.lang.String name, EventHandler <ActionEvent> onFinished, java.util.Collection <KeyValue <? >> values)

public KeyFrame(Duration time, java.lang.String name, EventHandler <ActionEvent> onFinished, KeyValue <?> ... values)

public KeyFrame(Duration time, EventHandler <ActionEvent> onFinished, KeyValue <?> ... values)

public KeyFrame(Duration time, java.lang.String name, KeyValue <?> ... values)

public KeyFrame(Duration time, KeyValue <?> ... values)

Зміну значення *JavaFX*-властивості представлено класом *javafx.animation.KeyValue*, екземпляр якого можна створити за допомогою конструкторів, що дозволяють встановити змінювану *JavaFX*-властивість, її кінцеве значення в результаті анімації і спосіб її зміни протягом анімації:

public KeyValue(WritableValue <T> target, T endValue, Interpolator <? super T> interpolator)

public KeyValue(WritableValue <T> target, T endValue)

Спосіб зміни значення *JavaFX*-властивості протягом анімації представлений класом *javafx.animation.Interpolator*, який має статичні поля:

Interpolator.DISCRETE – дискретна зміна значення *JavaFX*-властивості, при якому значення залишається початковим до закінчення тимчасового інтервалу, коли значення стає кінцевим;

Interpolator.LINEAR(за замовчуванням) – лінійна зміна значення *JavaFX*-властивості, при якому значення визначається за формулою

startValue +(endValue – startValue) fraction;

Interpolator.EASE_BOTH – використовується величина 0.2 для

приросту і зменшення значення *JavaFX*-властивості;

Interpolator.EASE_IN – використовується величина 0.2 для приросту значення *JavaFX*-властивості;

Interpolator.EASE_OUT – використовується величина 0.2 для зменшення значення *JavaFX*-властивості.

Крім того, можна створити окремий клас *Interpolator* з перевизначенням його методів, що описують зміну значення *JavaFX*-властивості.

Transition-анімація з вбудованою тимчасової шкалою також використовує об'єкт *Interpolator* як значення властивості *interpolator* класу *Transition*.

Таким чином, анімацію зміни значень декількох *JavaFX*-властивостей можна створити двома способами. Перший спосіб – це створення одного ключового кадру *KeyFrame* і додавання в нього декількох об'єктів *KeyValue*. Інший спосіб – це створення окремих ключових кадрів *KeyFrame* для кожного з об'єктів *KeyValue* і додавання їх в *Timeline*-анімацію.

Transition-анімація з вбудованою тимчасової шкалою, на відміну від *Timeline*-анімації, описує зміну в часі обмеженого набору *JavaFX*-властивостей, таких як прозорість, просторове положення, обертання і масштабування вузла графа сцени, а також колір заповнення і колір контуру форми *Shape*.

Анімація прозорості вузла графа сцени створюється за допомогою класу *FadeTransition*, що має властивості:

byValue – крок зміни властивості прозорості;

duration – тривалість анімації;

fromValue – початкове значення прозорості;

node – цільовий вузол графа сцени даної анімації;

toValue – кінцеве значення прозорості.

Примірник класу *FadeTransition* створюється за допомогою класу-фабрики *FadeTransitionBuilder* або за допомогою конструкторів

public FadeTransition(Duration duration, Node node)

public FadeTransition(Duration duration)

public FadeTransition()

Анімація просторового положення вузла графа сцени створюється за допомогою класів *PathTransition* і *TranslateTransition*.

Клас *PathTransition* дозволяє створювати переміщення графічного об'єкта уздовж кривої за допомогою властивостей:

duration – тривалість анімації;

orientation – орієнтація, якщо *PathTransition.OrientationType.NONE* – орієнтація графічного об'єкта не змінюється, якщо *PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT* – графічний об'єкт перпендикулярний відносно кривої свого переміщення;

path – об'єкт *javafx.scene.shape.Shape*, що представляє криву переміщення.

Примірник класу *PathTransition* створюється за допомогою класу-фабрики *PathTransitionBuilder* або за допомогою конструкторів:

public PathTransition()

public PathTransition(Duration duration, Shape path, Node node)

public PathTransition(Duration duration, Shape path)

Клас *TranslateTransition* дозволяє створювати переміщення графічного об'єкта з однієї 3D-точки в іншу 3D-точку за допомогою властивостей:

node – цільової вузол для анімації;

duration – тривалість анімації;

fromX – початкова координата переміщення по осі *x*;

fromY – початкова координата переміщення по осі *y*;

fromZ – початкова координата переміщення по осі *z*;

toX – кінцева координата переміщення по осі *x*;

toY – кінцева координата переміщення по осі *y*;

toZ – кінцева координата переміщення по осі *z*;

byX – крок переміщення по осі *x*;

byY – крок переміщення по осі *y*;

byZ – крок переміщення по осі *z*.

Примірник класу *TranslateTransition* можна створити за допомогою класу-фабрики *TranslateTransitionBuilder* або за допомогою конструкторів:

public TranslateTransition(Duration duration, Node node)

public TranslateTransition(Duration duration)

public TranslateTransition()

Анімація обертання вузла графа сцени створюється за допомогою класу *RotateTransition*, які мають властивості:

node – цільової вузол анімації;

duration – тривалість анімації;

axis – вісь обертання *javafx.geometry.Point3D*;

fromAngle – початковий кут обертання;

toAngle – кінцевий кут обертання;

byAngle – крок обертання.

Примірник класу *RotateTransition* створюється за допомогою класу-фабрики *RotateTransitionBuilder* або за допомогою конструкторів:

public RotateTransition(Duration duration, Node node)

public RotateTransition(Duration duration)

public RotateTransition()

Анімація масштабування вузла графа сцени створюється за допомогою класу *ScaleTransition*, що має властивості:

node – цільової вузол анімації;

duration – тривалість анімації;

fromX – початкове значення масштабування по осі *x*;

fromY – початкове значення масштабування по осі *y*;

fromZ – початкове значення масштабування по осі *z*;

toX – кінцеве значення масштабування по осі *x*;

toY – кінцеве значення масштабування по осі *y*;

toZ – кінцеве значення масштабування по осі *z*;

byX – крок масштабування по осі *x*;

byY – крок масштабування по осі *y*;

byZ – крок масштабування по осі *z*.

Примірник класу *ScaleTransition* можна створити за допомогою класу-фабрики

ScaleTransitionBuilder або за допомогою конструкторів:

public ScaleTransition(Duration duration, Node node)

public ScaleTransition(Duration duration)

public ScaleTransition()

Анімація кольору заповнення форми *Shape* створюється за допомогою класу *FillTransition*, що має властивості:

duration – тривалість анімації;

fromValue – початкове значення кольору;

shape – цільовий об’єкт *javafx.scene.shape.Shape* анімації;

toValue – кінцеве значення кольору.

Примірник класу *FillTransition* створюється за допомогою класу-фабрики *FillTransitionBuilder* або за допомогою конструкторів:

public FillTransition(Duration duration, Shape shape, Color fromValue, Color toValue)

public FillTransition(Duration duration, Color fromValue, Color toValue)

public FillTransition(Duration duration, Shape shape)

public FillTransition(Duration duration)

public FillTransition()

Анімація кольору контуру форми *Shape* створюється за допомогою класу *StrokeTransition*, що має властивості:

shape – цільовий об'єкт *javafx.scene.shape.Shape* для анімації;

duration – тривалість анімації;

fromValue – початковий колір контуру;

toValue – кінцевий колір контуру.

Примірник класу *StrokeTransition* створюється за допомогою класу-фабрики *StrokeTransitionBuilder* або за допомогою конструкторів:

public StrokeTransition(Duration duration, Shape shape, Color fromValue, Color toValue)

public StrokeTransition(Duration duration, Color fromValue, Color toValue)

public StrokeTransition(Duration duration, Shape shape)

public StrokeTransition(Duration duration)

public StrokeTransition()

Класи *ParallelTransition* і *SequentialTransition* дають можливість групувати анімації в паралельне і послідовне виконання.

Клас *ParallelTransition* має властивість *node*(цільової вузол графа сцени для анімації) і конструктори:

public ParallelTransition()

public ParallelTransition(Node node, Animation ... children)

public ParallelTransition(Animation ... children)

public ParallelTransition(Node node)

також клас-фабрику *ParallelTransitionBuilder*. Метод *getChildren().AddAll* дозволяє поповнити список дочірніх паралельних анімацій.

Клас *SequentialTransition* має властивість *node*(цільової вузол графа сцени для анімації) і конструктори:

public SequentialTransition()

public SequentialTransition(Node node, Animation ... children)

public SequentialTransition(Animation ... children)

public SequentialTransition(Node node)

також клас-фабрику *SequentialTransitionBuilder*. Метод *getChildren().AddAll* дозволяє поповнити список дочірніх послідовних анімацій.

Клас *PauseTransition* дозволяє зробити паузу в послідовності анімацій. Клас *PauseTransition* має властивість *duration*(тривалість паузи) і конструктори *public PauseTransition*(*Duration duration*) і *public PauseTransition*(), а також клас-фабрику *PauseTransitionBuilder*.

Клас *AnimationTimer* дозволяє створювати таймер, що викликається в кожному кадрі анімації. Створити таймер можна розширивши абстрактний клас *AnimationTimer* з перевизначенням його методу *handle*(*long now*), що викликається в кожному кадрі. Для управління таймером клас *AnimationTimer* пропонує методи *start*() і *stop*().

JavaFX-трансформація

Пакет *javafx.scene.transform* платформи *JavaFX* 2.0 забезпечує трансформації вузлів графа сцени, що складаються з афінних перетворень: обертань, переміщень, масштабування і зсуву.

На відміну від анімації, трансформації графічних об’єктів не мають плавного видимого переходу від початкової точки до кінцевої точки протягом певного проміжку часу, а виконуються відразу.

Базовим класом *JavaFX*-трансформацій є клас *javafx.scene.transform.Transform*, що має реалізації у вигляді класів *Affine*, *Rotate*, *Scale*, *Shear* і *Translate*.

Застосувати *JavaFX*-трансформації до вузла графа сцени можна двома способами. Перший спосіб – це використовувати метод *getTransforms*() класу *javafx.scene.Node*, який повертає список *ObservableList<Transform>* об’єктів *Transform*, поповнити який можна методом *addAll()*. Інший спосіб – це застосування методів *setRotate()*, *setRotationAxis()*, *setScaleX()*, *setScaleY()*, *setScaleZ()*, *setTranslateX()*, *setTranslateY()* і *setTranslateZ()* класу *javafx.scene.Node*, що забезпечують трансформації обертання, масштабування і переміщення для вузла графа сцени.

Клас *Affine* представляє афінне перетворення матриці:

mxx mxy mxz tx
myx myy myz ty
mzx mzy mzz tz

за допомогою властивостей:

mxx – *X*-множник матриці;
mxy – *XY*-множник матриці;
mxz – *XZ*-множник матриці;
tx – зсув по осі *x*;

myx – *YX*-множник матриці;
myy – *Y*-множник матриці;
myz – *YZ*-множник матриці;
ty – зсув по осі *y*;
mzx – *ZX*-множник матриці;
mzy – *ZY*-множник матриці;
mzz – *Z*-множник матриці;
tz – зсув по осі z.

Примірник класу *Affine* можна створити за допомогою класу-фабрики *AffineBuilder*, за допомогою конструктора *public Affine()* або за допомогою статичного методу *affine()* класу *Transform*, що повертає Affine-об'єкт.

Афінні перетворення відображають *n*-мірний об'єкт в *n*-мірний, зберігають паралельність ліній і площин, а також пропорції паралельних об'єктів.

За допомогою афінних перетворень можна створювати трансформації обертання, переміщення, масштабування і зсуву.

Клас *Rotate* забезпечує обертання вузла графа сцени за допомогою властивостей:

angle – кут обертання;
pivotX – горизонтальна координата опорної точки обертання;
pivotY – вертикальна координата опорної точки обертання;
pivotZ – *Z*-координата опорної точки обертання.

Примірник класу *Rotate* можна створити за допомогою класу-фабрики *RotateBuilder*, за допомогою набору конструкторів:

public Rotate(), public Rotate(double angle)
public Rotate(double angle, Point3D axis)
public Rotate(double angle, double pivotX, double pivotY)
public Rotate(double angle, double pivotX, double pivotY, double pivotZ)
public Rotate(double angle, double pivotX, double pivotY,
double pivotZ, Point3D axis)

або за допомогою статичного методу *rotate()* класу *Transform*, який повертає *Rotate*-об'єкт.

Клас *Scale* забезпечує масштабування вузла графа сцени за допомогою властивостей:

x – множник масштабування по осі *x*;
y – множник масштабування по осі *y*;
z – множник масштабування по осі *z*;

pivotX – горизонтальна координата опорної точки масштабування;

pivotY – вертикальна координата опорної точки масштабування;

pivotZ – *Z*-координата опорної точки масштабування.

Примірник класу *Scale* можна створити за допомогою класу-фабрики *ScaleBuilder*, за допомогою набору конструкторів:

public Scale()

public Scale(double x, double y)

public Scale(double x, double y, double pivotX, double pivotY)

public Scale(double x, double y, double z)

public Scale(double x, double y, double z, double pivotX, double pivotY, double pivotZ)

або за допомогою статичного методу *scale()* класу *Scale*-об’єкт.

Клас *Shear* забезпечує зміщення вузла графа сцени за допомогою властивостей:

x – множник по осі *x* від -1 до 1;

y – множник по осі *y* від -1 до 1;

pivotX – горизонтальна координата опорної точки зсуву;

pivotY – вертикальна координата опорної точки зсуву.

Примірник класу *Shear* можна створити за допомогою класу-фабрики *ShearBuilder*, за допомогою набору конструкторів:

public Shear()

public Shear(double x, double y)

public Shear(double x, double y, double pivotX, double pivotY)

або за допомогою статичного методу *shear()* класу *Transform*, що повертає Shear-об’єкт.

Клас *Translate* забезпечує переміщення вузла графа сцени за допомогою властивостей:

x – зміщення по осі *x*;

y – зміщення по осі *y*;

z – зміщення по осі *z*.

Примірник класу *Translate* можна створити за допомогою класу-фабрики *TranslateBuilder*, за допомогою набору конструкторів

public Translate()

public Translate(double x, double y)

public Translate(double x, double y, double z)

або за допомогою статичного методу *translate()* класу *Transform*, що повертає *Translate*-об’єкт.

Приклад створення JavaFX-додатку, які містить трансформацію та анімацію

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить трансформацію та анімацію, в меню *Файл* оберіть *Створити проект | Java with Ant | JavaFX | JavaFX Application*, натисніть кнопку *Далі*, введіть ім'я проекту *JavaFXApplicationTransition*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з'явиться код згенерованого класу *JavaFXApplicationTransition*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationTransition extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
```

```
    }
}
```

2. У методі *start()* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування трансформації та анімації” та зробіть видимим об’єкт Stage:

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.GRAY);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування трансформації та анімації");
primaryStage.show();
```

4. Продемонструйте *Transition*- і *Timeline*-анімацію букв тексту. *Transition*-анімація має складатись із паралельних анімацій переміщення, обертання і масштабування, а *Timeline*-анімація має змінювати розташування джерела світла для ефекту підсвічування тексту:

```
// Створення ефекту підсвічування тексту
final Light.Point lightPoint = new Light.Point();
lightPoint.setColor(Color.WHITE);
lightPoint.setX(0.0);
lightPoint.setY(0.0);
lightPoint.setZ(100.0);
final Lighting effect = new Lighting();
effect.setLight(lightPoint);
effect.setDiffuseConstant(1.5);
effect.setSpecularConstant(1.5);
effect.setSurfaceScale(8);
// Створення букв тексту
final Text tJ = new Text();
tJ.setEffect(effect);
tJ.setX(80);
tJ.setY(250);
tJ.setText( "J");
tJ.setFill(Color.RED);
tJ.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text ta1 = new Text();
ta1.setEffect(effect);
ta1.setX(120);
```

```
        ta1.setY(250);
        ta1.setText( "a");
        ta1.setFill(Color.RED);
        ta1.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text tv = new Text();
        tv.setEffect(effect);
        tv.setX(170);
        tv.setY(250);
        tv.setText( "v");
        tv.setFill(Color.RED);
        tv.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text ta2 = new Text();
        ta2.setEffect(effect);
        ta2.setX(220);
        ta2.setY(250);
        ta2.setText( "a");
        ta2.setFill(Color.RED);
        ta2.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text tF = new Text();
        tF.setEffect(effect);
        tF.setX(270);
        tF.setY(250);
        tF.setText( "F");
        tF.setFill(Color.RED);
        tF.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text tX = new Text();
        tX.setEffect(effect);
        tX.setX(320);
        tX.setY(250);
        tX.setText( "X");
        tX.setFill(Color.RED);
        tX.setFont(Font.font(null, FontWeight.BOLD, 80));
        // Створення анімацій переміщення, обертання і масштабування
букв тексту
        final TranslateTransition ttJ = new
TranslateTransition(Duration.millis(1000), tJ);
        ttJ.setCycleCount(2);
        ttJ.setByX(-200.0);
        ttJ.setToY(-270.0);
```

```
        ttJ.setAutoReverse(true);
        final RotateTransition rtJ = new
RotateTransition(Duration.millis(500), tJ);
        rtJ.setByAngle(360);
        rtJ.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtJ.setCycleCount(4);
        final ScaleTransition stJ = new
ScaleTransition(Duration.millis(1000), tJ);
        stJ.setByX(-1.5);
        stJ.setByY(-1.5);
        stJ.setCycleCount(2);
        stJ.setAutoReverse(true);
        final TranslateTransition tta1 = new
TranslateTransition(Duration.millis(1000), ta1);
        tta1.setCycleCount(2);
        tta1.setByX(-100.0);
        tta1.setToY(-270.0);
        tta1.setAutoReverse(true);
        final RotateTransition rta1 = new
RotateTransition(Duration.millis(500), ta1);
        rta1.setByAngle(360);
        rta1.setAxis(new Point3D(10.0, 10.0, 10.0));
        rta1.setCycleCount(4);
        final ScaleTransition sta1 = new
ScaleTransition(Duration.millis(1000), ta1);
        sta1.setByX(-1.5);
        sta1.setByY(-1.5);
        sta1.setCycleCount(2);
        sta1.setAutoReverse(true);
        final TranslateTransition ttv = new
TranslateTransition(Duration.millis(1000), tv);
        ttv.setCycleCount(2);
        ttv.setByX(0.0);
        ttv.setToY(-270.0);
        ttv.setAutoReverse(true);
        final RotateTransition rtv = new
RotateTransition(Duration.millis(500), tv); rtv.setByAngle(360);
        rtv.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtv.setCycleCount(4);
```

```
        final ScaleTransition stv = new
ScaleTransition(Duration.millis(1000), tv); stv.setByX(-1.5);
        stv.setByY(-1.5);
        stv.setCycleCount(2);
        stv.setAutoReverse(true);
        final TranslateTransition tta2 = new
TranslateTransition(Duration.millis(1000), ta2);
        tta2.setCycleCount(2);
        tta2.setByX(100.0);
        tta2.setToY(-270.0);
        tta2.setAutoReverse(true);
        final RotateTransition rta2 = new
RotateTransition(Duration.millis(500), ta2); rta2.setByAngle(360);
        rta2.setAxis(new Point3D(10.0, 10.0, 10.0));
        rta2.setCycleCount(4);
        final ScaleTransition sta2 = new
ScaleTransition(Duration.millis(1000), ta2); sta2.setByX(-1.5);
        sta2.setByY(-1.5);
        sta2.setCycleCount(2);
        sta2.setAutoReverse(true);
        final TranslateTransition ttF = new
TranslateTransition(Duration.millis(1000), tF);
        ttF.setCycleCount(2);
        ttF.setByX(200.0);
        ttF.setToY(-270.0);
        ttF.setAutoReverse(true);
        final RotateTransition rtF = new
RotateTransition(Duration.millis(500), tF); rtF.setByAngle(360);
        rtF.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtF.setCycleCount(4);
        final ScaleTransition stF = new
ScaleTransition(Duration.millis(1000), tF);
        stF.setByX(-1.5);
        stF.setByY(-1.5);
        stF.setCycleCount(2);
        stF.setAutoReverse(true);
        final TranslateTransition ttX = new
TranslateTransition(Duration.millis(1000), tX);
        ttX.setCycleCount(2);
```

```
        ttX.setByX(300.0);
        ttX.setToY(-270.0);
        ttX.setAutoReverse(true);
        final RotateTransition rtX = new
RotateTransition(Duration.millis(500), tX);
        rtX.setByAngle(360);
        rtX.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtX.setCycleCount(4);
        final ScaleTransition stX = new
ScaleTransition(Duration.millis(1000), tX);
        stX.setByX(-1.5);
        stX.setByY(-1.5);
        stX.setCycleCount(2);
        stX.setAutoReverse(true);
        // Групування анімацій в паралельне виконання
        final ParallelTransition ptJ = new ParallelTransition(tJ, ttJ, rtJ, stJ);
        final ParallelTransition pta1 = new ParallelTransition(ta1, tta1, rta1,
sta1);
        final ParallelTransition ptv = new ParallelTransition(tv, ttv, rtv, stv);
        final ParallelTransition pta2 = new ParallelTransition(ta2, tta2, rta2,
sta2);
        final ParallelTransition ptF = new ParallelTransition(tF, ttF, rtF, stF);
        final ParallelTransition ptX = new ParallelTransition(tX, ttX, rtX,
stX);
        // Створення анімації переміщення джерела світла ефекту
підсвічування тексту
        final Timeline timeline = new Timeline();
        timeline.setCycleCount(2);
        timeline.setAutoReverse(true);
        KeyValue kv = new KeyValue(lightPoint.xProperty(), 500.0,
Interpolator.EASE_BOTH);
        KeyFrame kf = new KeyFrame(Duration.millis(1000), kv);
        timeline.getKeyFrames(). addAll(kf);
        timeline.setDelay(Duration.millis(2000));
        // Створення кнопки запуску анімації
        Button btn = new Button();
        btn.setLayoutX(10);
        btn.setLayoutY(450);
        btn.setText( "Анімація");
```

```
btn.setStyle("- fx-font: bold italic 14pt Georgia; -fx-text-fill: white; -fx-background-color: gray; -fx-border-width: 1px; -fx-border-color: white ");
btn.setOnAction((ActionEvent event) -> {
    if(! timeline.getStatus().equals(Animation.Status.RUNNING)) {
        ptJ.play();
        pta1.play();
        ptv.play();
        pta2.play();
        ptF.play();
        ptX.play();
        timeline.play(); }
});
```

4. Створені об’єкти додайте в кореневий вузол графа сцени:

```
root.getChildren().addAll(btn, tJ, ta1, tv, ta2, tF, tX);
```

5. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити *Transition*- і *Timeline*-анімацію букв тексту (рис. 1).

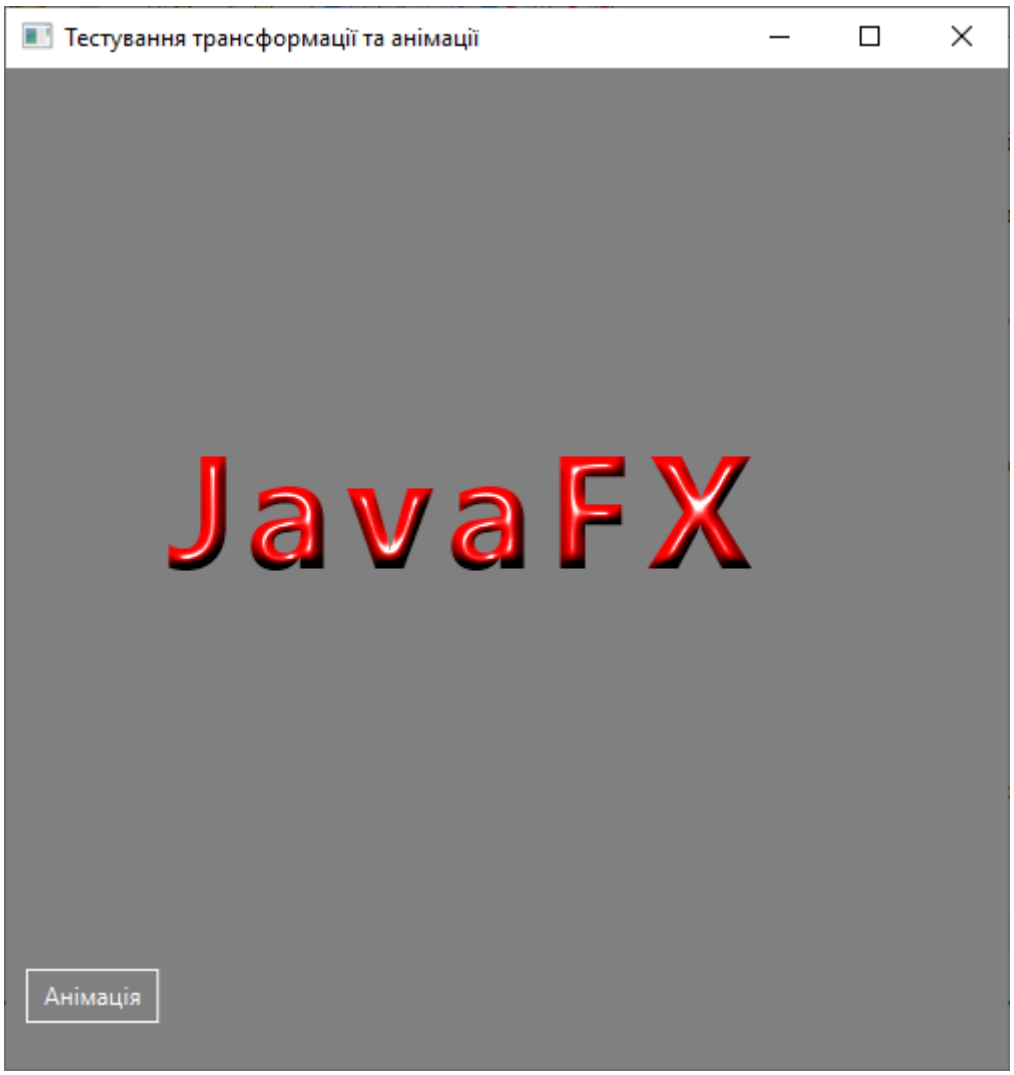

Рис. 1. JavaFX-додаток, що демонструє *Transition*- і *Timeline*-анімацію букв тексту

Контрольні запитання

1. Яке призначення *Transition*- і *Timeline*-анімації?
2. Як створити *Transition*- і *Timeline*-анімацію?
3. Які основні властивості мають *Transition*- і *Timeline*-анімації?

Список використаних джерел

1. Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. СПб.: БХВ-Петербург, 2012. 320 с.
2. Муляр В. П. Основи розробки додатків з використанням технології JavaFX. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2018. Вип. № 30-31. С. 104–110.
3. Муляр В. П. Розробка JavaFX-додатків із використанням Scene Builder. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2020. Вип. № 39. С. 181–189.
4. Муляр В. П., Яцюк С. М. Елементи комп'ютерної графіки у візуалізації результатів моделювання фізичних явищ і процесів. Комп'ютерно-орієнтовані технології: освіта, наука, виробництво. 2016. № 23. С. 80–84.
5. Спірінцева О. В., Литвинов О. А., Герасимов В. В. Java-технології та мобільні пристрої. Алгоритми і структури даних: навч. посіб. Д.: Вид-во ДНУ ім. О. Гончара, 2016. 140 с.
6. Java Tutorial. URL: https://www.w3schools.com/java/default.asp
7. Java. Классы. Объектно-ориентированное программирование. URL: https://metanit.com/java/tutorial/3.1.php
8. Підручник з Java. URL: https://www.javatpoint.com/java-tutorial
9. GDB online Debugger / Compiler. URL: https://www.onlinegdb.com/
10. Java – Учебники по программированию. URL: https://betacode.net/
11. Apache NetBeans. URL: https://netbeans.apache.org/download/index.html
12. Java Course. URL: http://java-course.ru/begin/introduce/
13. Java SE Downloads. URL: https://www.oracle.com/java/technologies/javase-downloads.html
14. JavaFX. URL: https://gluonhq.com/products/javafx/
15. Scene Builder. URL: https://gluonhq.com/products/scene-builder/
16. Введение в Java FX. URL: https://metanit.com/java/javafx/1.1.php
17. Руководство JavaFX для начинающих – Hello JavaFX. URL: https://betacode.net/10623/javafx-tutorial-for-beginners

Навчальне видання

Муляр Вадим Петрович

ПРОЄКТУВАННЯ І РОЗРОБКА КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

Конспект лекцій