Практичне заняття №1

Тема: Графічні інтерфейси користувача в Java.

Мета: Вивчити особливості роботи з інтерфейсами у JavaFX.

Завдання: Здобути навички створення та відлагодження проектів обробки даних з використанням графічних інтерфейсів користувача на JavaFX.

(max: 5 балів)

Теоретичні відомості:

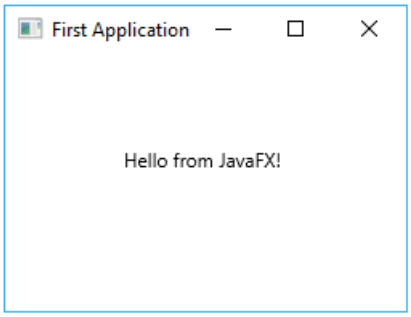
Однією з найпопулярніших технологій створення GUI-додатків є JavaFX. JavaFX представляє інструментарій для створення кросплатформових графічних додатків мовою Java. За допомогою JavaFX можна створювати програми для різних операційних систем: Windows, MacOS, Linux і для найрізноманітніших пристроїв: десктопи, смартфони, планшети, вбудовані пристрої, ТВ.

Серед особливостей JavaFX слід зазначити хардверні прискорення, створення GUI за допомогою CSS і XML (FXML), можливість використовувати контроли JavaFX в Swing, а також набір нових красивих контролів, в тому числі для малювання діаграм і 3D.

Найпростіша програма для JavaFX:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.*;
public class Main extends Application{
    public static void main(String[] args) {  Application.launch(args);  }
   @Override  //stage представляє графічний інтерфейс
   public void start(Stage stage) {
      Text text = new Text("Hello from JavaFX!");//визначення напису
      text.setLayoutY(80); // задання положення напису по осі Y
      text.setLayoutX(100); // задання положення напису по осі X
      Group group = new Group(text); // створення групи елементів
      Scene scene = new Scene(group); //створення сцени
      stage.setScene(scene); //визначення сцени для об'єкта stage
      stage.setTitle("First Application"); // задання заголовку
      stage.setWidth(300); // задання ширини вікна
      stage.setHeight(250); // задання висоти вікна
      stage.show();
   } // scene – контейнер верхнього рівня
}
```

Результат:



Головний класс додатку JavaFX повинен успадковуватись від класу *javafx.application.Application*. Об'єкт цього класу проходить ряд етапів життєвого циклу. Всі ці етапи представлені **методами класу *Application***, які викликаються автоматично середовищем JavaFX:

• *init()* ініцілізує додаток до його запуску; метод не повинен використовуватися для створення

графічного інтерфейсу або окремих його частин;

- *start(Stage stage)* призначений для визначення в ньому графічного інтерфейсу;
- stop() викликається після закриття програми, наприклад, після того, як користувач натиснув на хрестик в правому верхньому куті.

Процес роботи додатку:

1. Запуск виконуючого середовища JavaFX.
2. Виклик конструктора класу, який розширює клас *Application*.
3. Виклик методу *init()*.
4. Виклик методу *start(javafx.stage.Stage)*, в який середовище JavaFX передає створений об'єкт *Stage*. Таким чином, додаток починає працювати.
5. Очікування закриття додатку.
6. При завершенні роботи додатку середовище викликає методу *stop()*.

Всі ці методи можна перевизначити в головному класі програми. Але, як правило, визначається один метод *start()*, в якому визначаються налаштування графічного інтерфейсу.

Основою для створення графічного інтерфейсу в JavaFX є **клас *javafx.stage.Stage***. Фактично, він є контейнером, в який поміщаються всі інші компоненти інтерфейсу. Його конкретна реалізація залежить від платформи, на якій запускається додаток. Так, на десктопах це буде окреме графічне вікно, а на мобільних пристроях інтерфейс може представляти весь екран пристрою. Використовуючи свої методи, *Stage* управляє позиціонуванням, розмірами і деякими іншими налаштуваннями вікна програми.

Клас *javafx.scene.Scene* представляє контейнер для всіх графічних елементів всередині об'єкта Stage у вигляді графа, який називається *Scene Graph*. Всі вузли цього графа, тобто всі вкладені елементи повинні представляти **клас *javafx.scene.Node***. Але **кореневої вузол** цього графа повинен представляти об'єкт класу, який успадковується від ***javafx.scene.Parent***. Parent – це, фактично, контейнер, що може містити інші елементи.

Для визначення **кореневого вузла** в Scene використовується один з **конструкторів об'єкта Scene**. Основні з них:

- *Scene(Parent root)* створює *Scene* з кореневим вузлом *root*;
- *Scene(Parent root, double width, double height)* створює *Scene* з кореневим вузлом *root*, з шириною *width* і висотою *height*;
- *Scene(Parent root, Paint fill)* створює *Scene* з кореневим вузлом *root* і задає фоновий колір;
- *Scene (Parent root, double width, double height, Paint fill)* створює *Scene* з кореневим вузлом *root*, з шириною *width* і висотою *height* і задає фоновий колір.

При цьому всі конструктори приймають в якості першого параметра кореневий вузол, тому при створенні об'єкта *Scene* доведеться визначити і кореневий вузол. Розглянемо код програми

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
public class Main extends Application {
    public static void main (String [] args) {
        Application.launch (args);
    }
    @Override
    public void start (Stage stage) {
        Label label = new Label( "Hello"); // текстова мітка
        Button button = new Button( "Button"); // кнопка
        Group group = new Group(button); // вкладений вузол Group
```

```
        FlowPane root = new FlowPane(label, group); // кореневий вузол
        Scene scene = new Scene(root, 300, 150); // створення Scene
        stage.setScene(scene); // задання Scene для Stage
        stage.setTitle("Hello JavaFX");
        stage.show();
    }
}
```

Тут в якості кореневого вузла *root* виступає клас *FlowPane*, який розташовує вкладені елементи підряд і який, як і *Group*, успадкований від класу *Parent*.*FlowPane* містить об'єкт *Label* (текстова мітка) і об'єкт *Group*. А об'єкт *Group*, в свою чергу, містить об'єкт *Button* (кнопка).

Для взаємодії з користувачем в JavaFX використовується **модель подій**. У цій моделі є джерело події – деякий елемент управління (кнопка, …), який генерує подію, і є один або кілька слухачів або обробників події, які виконуються при настанні події. Коли елемент управління генерує подію, то обробник обробляє цю подію.

Сцена програми та її граф показані на рисунку 1.1.:

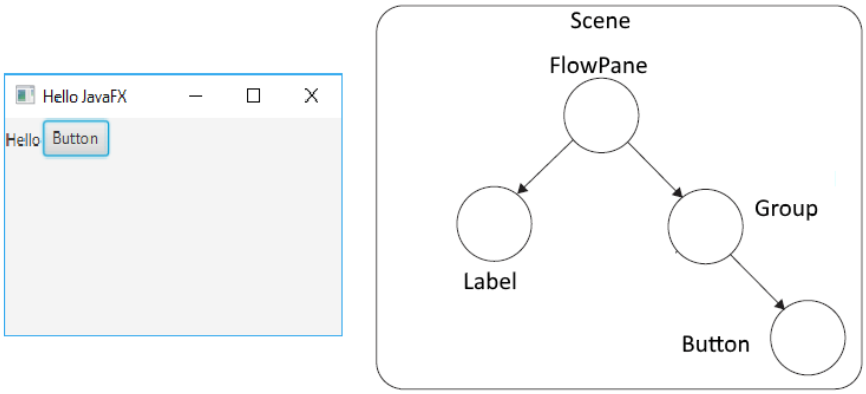


Рис. 1.1. Сцена та її граф

Розглянемо базовий приклад обробки подій:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
public class Main extends Application {
    public static void main (String [] args) { Application.launch(args); }
    @Override
    public void start (Stage stage){
        Button btn = new Button();
        btn.setText("Потрібно натиснути!");
        btn.setOnAction(new EventHandler <ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                btn.setText("Ви натиснули на кнопку!");
            }
        });
        Group root = new Group(btn);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Hello JavaFX");
        stage.setWidth(250);
```

```
        stage.setHeight(200);
        stage.show();
    }
}
```

Тут створено кнопку, яка представлена об'єктом класу *Button*. При натисканні на кнопку виникає подія натискання, і ця подія обробляється.

Базовим класом для всіх подій є **клас *javafx.event.Event***, який успадкований від класу *java.util.EventObject*. При генерації події, наприклад, при натисканні на кнопку, створюється **об'єкт *Event***, через який передається інформація про подію. В даному випадку при натисканні на кнопку буде генеруватися подія *muny javafx.event.ActionEvent*.

Для визначення обробника події використовується функціональний **інтерфейс *EventHandler***.

```
public interface EventHandler <T extends Event>
{ void handle (T event); }
```

Інтерфейс *EventHandler* типізується типом, який успадкований від класу *Event* і який, фактично, є подією. У нашому випадку це клас *ActionEvent*.

Для прикріплення обробника події *EventHandler* до події елемента управління застосовується метод *btn.setOnAction()*, в який передається реалізація інтерфейсу *EventHandler*. У методі *handle* визначаються дії, які будуть виконуватися при натисканні на кнопку:

```
btn.setOnAction (new EventHandler <ActionEvent> () {
    @Override
    public void handle(ActionEvent event) {
        btn.setText("Ви натиснули на кнопку!");
    }
});
```

В обробнику вище змінюється надпис на кнопці. Часто замість визначення явної реалізації інтерфейсу *EventHandler* застосовуються лямбда-вирази. Наприклад, можна переписати метод *btn.setOnAction()* наступним чином:

```
btn.setOnAction(e->{btn.setText("Ви натиснули на кнопку!"); });
```

Для спрощення організації та роботи з інтерфейсом в JavaFX використовується мова розмітки FXML, яка створена на основі XML. FXML дозволяє визначити інтерфейс програми декларативним чином подібно до того, як веб-сторінки визначаються за допомогою HTML.

Інтерфейс програми також можна визначити за допомогою **програми Scene Builder**. З допомогою останньої, користувацький інтерфейс (UI) створюється шляхом перетягування елементів управління з палітри. Ця інформація зберігається у файлі FXML.

До елементів управління в JavaFX належать: кнопки та мітки, CheckBox, ToogleButton, RadioButton, текстові елементи управління, ScrollPane, Slider, ListView, ComboBox, ChoiceBox, TreeView, ToolTip, TableView.

Індивідуальні завдання:

(Для виконання індивідуальних завдань № варінта є для 532 групи порядковим номером прізвища студента в списку групи, для 531 – вказаний у додатку (окремий документ). Усі проекти та, за наявності, тести до них завантажити у власні репозиторії на Git Hub. Посилання на репозиторії проектів вказати у звіті та обов'язково долучати скріни успішного виконання)

Написати программу для створення GUI-додатку до задачі згідно вашого варіанту. Для створення GUI-додатку використати засоби бібліотеки JavaFX.

(5 балів)

№	Завдання
1.	Алгоритм, який повертає значення 1, якщо точка з координатами (x,y) потрапляє всередину прямокутника з координатами точок головної діагоналі (x1,y1)-(x2,y2), та 0 - в іншому випадку
2.	Алгоритм, який обчислює площу трикутника зі сторонами a, b, c.
3.	Алгоритм, який повертає значення 1, якщо відрізки [a,в] та [c,d] мають точки перетину, і 0 - в іншому випадку.
4.	Алгоритм, який повертає значення 1, якщо із чисел x,y,z тільки два рівні між собою, і 0 - в іншому випадку.
5.	Алгоритм, який визначає, чи точка (x,y) лежить поза колом радіуса r з центром в точці (1,0).
6.	Алгоритм для перевірки того, чи значення x не належить відрізку [0,1].
7.	Алгоритм для перевірки того, чи значення x належить відрізку [2,5] або [-1,1].
8.	Алгоритм, який перевіряє, чи точка з координатами (x,y) потрапляє всередину кола радіуса R з центром у початку координат.
9.	Алгоритм, який визначає чи відрізки [a,в] та [c,d] мають спільні точки.
10.	Алгоритм для обчислення виразу $x = \sqrt{a - b^2} + \sqrt{ab}$, для цілих додатних змінних a,b.
11.	Алгоритм для обчислення значення виразу $a = \sqrt{x^2 y - 14z}$. Для додатних цілих змінних x,y,z
12.	Алгоритм, який повертає 1, якщо рівняння $ax^2 + bx + c = 0$ (a<>0, b<>0, c<>0) має два різних дійсних корені, і 0 - в інших випадках.
13.	Алгоритм для визначення того, чи задане ціле число є парним.
14.	Алгоритм, який серед заданих трьох чисел a, b, c знаходить мінімальне.
15.	Алгоритм для визначення відстані між точками з координатами (x1,y1) та (x2,y2).
16.	Алгоритм, який повертає значення 1, якщо точка з координатами (x,y) потрапляє всередину трикутника з координатами точок головної вершин (x1,y1), (x2,y2), (x3,y3) та 0 - в іншому випадку.
17.	Алгоритм, який обчислює площу еліпса з радіусами a, b, та з центром у початку координат.
18	Алгоритм, який повертає значення 1, якщо точка з координатами (x,y) потрапляє всередину шестикутника з координатами точок вершин та 0 - в іншому випадку.
19	Алгоритм для визначення того, чи задане ціле число є непарним.
20	Алгоритм для перевірки того, чи значення x належить відрізку [-2,3].
21	Алгоритм, який повертає значення 1, якщо із чисел x,y,z всі рівні між собою, і 0 - в іншому випадку.
22	Алгоритм, який серед заданих трьох чисел a, b, c знаходить максимальне.
23	Алгоритм, який визначає, чи точка (x,y) лежить у колі радіуса r з центром в точці (1,1).
24	Алгоритм, який обчислює периметр трикутника зі сторонами a, b, c.
25	Алгоритм, який обчислює площу ромба з діагоналями a, b.