Крос-платформне програмування

ЛАБОРАТОРНА РОБОТА № 5

Основні принципи об’єктно-орієнтованого програмування в Java.

Створення візуального UI з використанням Java Swing.

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи
.

Львів

2024

Мета роботи: Ознайомлення з сценарієм збирання Java-проєктів Maven. Вивчення засад об'єктно-орієнтованого програмування та їх реалізації в Java. Знайомство з бібліотекою візуальних компонентів Java Swing та їх використанням для створення графічного інтерфейсу користувача.

Завдання до лабораторної роботи.

1. Реалізувати запропоновану у варіанті завдання ***ієрархію класів*** для зберігання даних створюваної програми.
2. Програма повинна забезпечувати зберігання колекції об'єктів в оперативній пам'яті, можливість додавання до колекції нових об'єктів, редагування та вилучення доданих раніше об'єктів.
3. Дані програми (колекція об'єктів) між сеансами роботи повинна зберігатися у ***файлі***.
4. Для взаємодії програми з користувачем реалізувати графічний віконний інтерфейс на базі візуальних компонентів з вбудованої бібліотеки ***Java Swing***.
5. Оформити письмовий звіт про виконання роботи.

Оформити письмовий звіт про виконання роботи.

Звіт повинен містити:

- титульний аркуш:
- умову задачі відповідного варіанту;
- код репозиторію для зберігання даних на Java;
- малюнки з зображенням конструкції вікон, що використовуватимуться для взаємодії користувача з програмою;
- код обробників подій візуальних компонентів на Java;
- знімки екрану з результатами роботи створеної системи для кожного з елементів реалізованого функціоналу;

*** За бажанням, додатково можна реалізувати аналогічний проєкт на Kotlin.

Короткі теоретичні відомості

Apache Maven — це засіб автоматизації роботи з програмними проєктами, який спочатку використовувався для Java-проєктів. Використовується для управління (management) та складання (build) програм. Створений 2002 року Джейсоном ван Зилом. За принципами роботи кардинально відрізняється від Apache Ant, та має простіший вигляд щодо build-налаштувань, яке надається в форматі XML.

XML-файл описує проєкт, його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах.

Раніше Maven був частиною *Jakarta Project*.

- Використовується чітка структура папок.
- Процес збирання записується в форматі xml в файлі pom.xml
- В IDEA можна додати підтримку Maven.

Життєвий цикл додатку на Apache Maven:

validate - перевіряє коректність метаінформації про проект

compile - компілює вихідні

test - проганяє тести класів з попереднього кроку

package - упаковує скомпільовані класи в jar або war

integration-test - інтеграційного тестування і проганяє тести

verify - перевіряє коректність пакета і задоволення вимог якості

install - заганяє пакет в локальний репозиторій, звідки він (пакет) буде доступний для використання як залежність в інших проектах

deploy - відправляє пакет на віддалений production сервер, звідки інші розробники його можуть отримати і використовувати

project – кореневий елемент файлу

dependencies – розділ підключення сторонніх бібліотек

plugins – розділ підключених плагінів

build – розділ збирання проектів, включає розділ плагінів і команди зборки проекту.

Java Swing

Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Це частина бібліотеки базових класів Java (JFC, Java Foundation Classes).

Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Компоненти Swing підтримують специфічні look-and-feel[en] модулі, що динамічно підключаються. Завдяки їм можлива емуляція

графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є відносно повільна робота, хоча останнім часом це не вдалося підтвердити через зростання потужності персональних комп'ютерів. Позитивна сторона — універсальність інтерфейсу створених програм на всіх

На початку існування Java класів Swing не було взагалі. Через слабкі місця в AWT (початковій GUI системі Java) було створено Swing. AWT визначає базовий набір елементів керування, вікон та діалогів, які підтримують придатний до використання, але обмежений у можливостях графічний інтерфейс. Однією з причин обмеженості AWT є те, що AWT перетворює свої візуальні компоненти у відповідні їм еквіваленти, що не залежать від платформи, які називаються рівноправними компонентами. Це означає, що зовнішній вигляд компонентів визначається платформою, а не закладається в Java. Оскільки компоненти AWT використовують «рідні» ресурси коду, вони називаються ваговитими (англ. highweigh).

Використання «рідних» рівноправних компонентів породжує деякі проблеми. По-перше, у зв'язку із різницею, що існує між операційними системами, компонент може виглядати або навіть вести себе по-різному на різноманітних платформах. Така мінливість суперечила філософії Java: «написане один раз, працює скрізь». По-друге, зовнішній вигляд кожного компонента був фіксованим (оскільки усе залежало від платформи), і це неможливо було змінити (принаймні, це важко було зробити). По-третє, використання ваговитих компонентів тягнуло за собою появу нових обмежень. Наприклад, ваговитий компонент завжди має прямокутну форму і є непрозорим.

Незабаром після появи початкової версії Java, стало очевидним, що обмеження, властиві AWT, були настільки незручними, що потрібно було знайти кращий підхід. У результаті з'явились класи Swing як частина бібліотеки базових класів Java (JFC). В 1997 році вони були включені до Java 1.1 у вигляді окремої бібліотеки. А починаючи з версії Java 1.2, класи Swing (а також усі останні, що входили до JFC) стали повністю інтегрованимими у Java.

Варіанти завдань.

1. Розробити систему для керування різними типами працівників (наприклад, штатних, неповний робочий день, підрядників) за допомогою успадкування. Кожен тип працівника повинен мати унікальні властивості та методи для розрахунку робочого часу оплати праці іт.п. Реалізувати фільтр працівників за типом.

2. Створити інформаційну систему агенства з оренди нерухомості з різними типами приміщень (наприклад, житлові квартири, котеджі, торгівельні площі, промислова нерухомість), що успадковуються від базового класу.

3. Розробитти програму фітнес-трекера, де користувачі можуть реєструвати різні типи активності (наприклад, біг, їзда на велосипеді, плавання), причому кожна діяльність успадковується від базового класу активності.

4. Створити ієрархію класів для зброї в комп'ютерній грі. (базовий клас «Зброя», похідні класи «Меч», «Спис», «Арбалет»). На основі створеної розробити програму зі списками персонажів для декількох гравців, що беруть участь в грі.

5. Створити систему для керування різними типами носіїв інформації в бібліотеці (наприклад, книга, DVD, журнал) за допомогою успадкування. Впровадити методи позичання, повернення та пошуку інформаційних засобів.

6. Розробити систему для зарахування студентів на різні напрямки навчання (наприклад, розробка ПЗ, адміністрування, кібербезпека, тощо), причому кожен курс успадковує базовий клас курсу, та має свої характеристики щодо навчальних предметів, способу відвідування занять, тривалості навчання і т.д.

7. Розробити спрощену соціальну медіа-платформу, де користувачі зможуть публікувати різні типи вмісту (наприклад, текст, зображення, відео), причому кожен тип вмісту успадковує базовий клас Post.

8. Створити базовий клас «Клієнт» та підкласи, «Роздрібний покупець», «Оптовий покупець», «Компанія-партнер», із спеціальними методи розрахунку за надані послуги та товари. Розробити інформаційну систему зі списком клієнтів деякої компанії.

9. Створити інформаційну систему для оренди різних типів транспортних засобів (наприклад, легкових автомобілів, вантажівок, спецтехніки) з різними орендними ставками та доступністю. Реалізувати пошук потрібного засобу та фільтацю за вартістю послуг.

10. Розробити інформаційну систему ветеринарної клініки для для запису натприйом домашніх тварин різних видів (собаки, кішки, черепахи, тощо), наоснові базового класу Pet.

11. Створити програму для керування меню ресторану з різними категоріями страв (наприклад, закуски, основні страви, десерти) за допомогою успадкування.

12. Розробити ієрархію видів транспортних засобів для комп’ютеної гри. Розробити методи та властивості індивідуальні для кожного виду. Реалізувати опрацювання списку гравців, з відображенням характеристики його транспортного засобу та можливістю заміни його на інший відповідно до правил гри.

13. Створити інформаційну систему для планування медичних прийомів із різними типами надавачів медичних послуг (наприклад, сімейний лікар, медсестра, лікар-спеціаліст і т. п.), які успадковують базовий клас «медик».

14. Створити клас ”Книга” з полями “назва”, ”автор”, “кількість сторінок”, “рік видання”. Розробит похідні класи `Художня книга`, `Підручник`, `Наукова книга` з додатковими полями та методами. На основі створеної ієрархії розробити ІС «Бібліотека» з надання доступу читачам до тієї чи іншої книги.

15. Створити ієрархію класів для персонажів комп'ютерної гри. (наприклад, базовий клас «Персонаж», похідні класи «Воїн», «Мольфар», «Хоракгерник»). На основі створеної розробити програму зі списками персонажів для декількох гравців, що беруть участь в грі.

16. Створити базовий клас «Платник податків» та підкласи, «Найманий працівник», «ФОП», «Юридична особа», із спеціальними методи розрахунку податку на основі джерела доходу. Розробити інформаційну систему зі списком платників податків деякої адміністративної території.

17. Створити інформаційну систему для бронювання рейсів із різними типами рейсів (наприклад, внутрішні, міжнародні, бізнес-клас), що успадковуються від базового класу рейсу.

18. Розробити систему обліку заробітної платні для різних посад працівників компанії за допомогою успадкування. Реалізувати методи розрахунку зарплати відповідно до посади та формування відомостей на оплату.

19. Створіть систему для керування різними типами банківських рахунків (наприклад, SavingsAccount, CheckingAccount, InvestmentAccount), реалізувавши відповідну ієрархію класів. Імплементувати методи для внесення, зняття та запиту балансу для кожного типу рахунку.

20. Розробити ієрархію матеріалів блогу різних типів (наприклад, категорія, тег, публікація, коментар), кожен тип матеріалу успадковує базовий клас Post, але має власний набір властивостей та методів відображення. На основі створеної ієрархії створити систему керування матеріалами блогу.

21. Створити ієрархію транспортних засобів (наприклад, автомобіль, вантажівка, мотоцикл). Розробити методи розрахунку потреб у паливі на заданий пробігу, вартості 1 км. проїзду та ін. Реалізувати пошук ТЗ за заданими параметрами.

22. Розробити програму-кошик для онлайн-покупок із різними типами товарів. Впровадити методи додавання товарів у кошик, розрахунку загальної вартості та друкування рахунку на оплату.

23. Розробити інформаційну систему рекрутингового агенства з різними типами вакансій (наприклад, повна зайнятість, часткова зайнятість, фріланс і т.п.) за використанням ієрархії відповідних класів. Кожен тип вакансії повинен мати унікальні властивості та методи для розрахунку оплати праці і т.п. Реалізувати пошук вакансій за спеціалізацією та типами.

24. Розробити ІС з інформацією про види тварин на планеті, використовуючи загальні класи живих організмів (наприклад, ссавці, птахи, риби і т.п.). Реалізувати методи для опису поведінки, специфічної для кожного виду тварин. Надати можливість користувачеві переглядати записи про види тварин кожного класу окремо.

25. Розробити систему для керування інформацією про співробітників (наприклад, ім’я, вік, відділ) із різними типами працівників (наприклад, повний робочий день, неповний робочий день, тимчасовий) за допомогою успадкування.

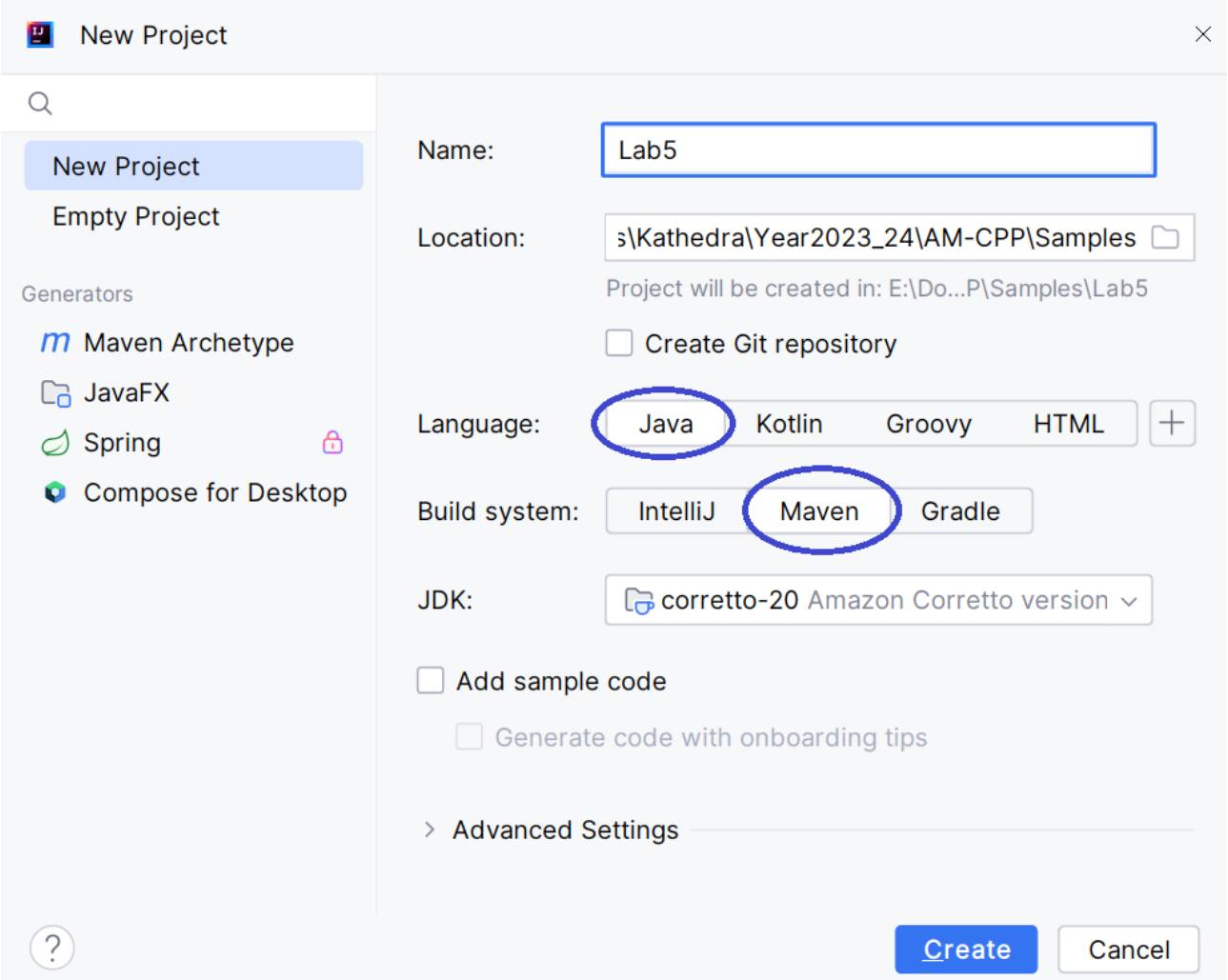
26. Створити ієрархію геометричних фігур (наприклад, коло, прямокутник, відрізок), де кожна фігура є підкласом базового класу фігур. Розробити програму для опрацювання «векторного малюка» у вигляді списку фігур різних розмірів та кольору.

Приклад розв'язування задачі 30

26. Створити ієрархію геометричних фігур (наприклад, коло, прямокутник, відрізок), де кожна фігура є підкласом базового класу фігур. Розробити програму для опрацювання «векторного малюка» у вигляді списку фігур різних розмірів та кольору.

Частина 1. Створення проєкту Java

Запускаємо IntelliJ IDEA. Створюємо новий проєкт з використанням мови програмування Java та сценарію Maven.



Необхідні бібліотеки класів додаватимемо у файл ***pom.xml*** за потребою. Бібліотека Java Swing міститься в усіх актуальнихверсіях пакету JDK. Пересвідчитися в цьому можна, створивши головний клас (наприклад Main) з таким кодом:

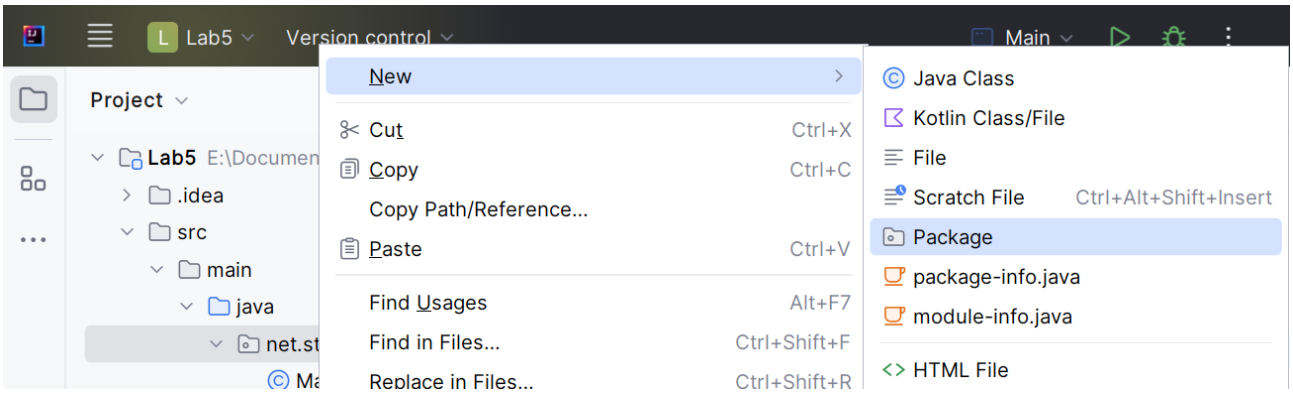
```
import javax.swing.*;

public class Main {
```

```
    public static void main(String[] args) {
        // вивід вікна з повідомленням
        JOptionPane.showMessageDialog(
                null,"Hello Swing!");
    }
}
```

Запуск цього коду має відобразити на екрані вікно з вказаним повідомленням.

Сценарій збирання Maven передбачає певну структуру папок проєкту. Файли з кодом проєкту слід розміщувати в папці ***src/main/java/***, а юніт тести за аналогічним маршрутом в папці ***src/test/java/***.

Ствримо пакет для проєкту (наприклад ***net.starbasic.minipaint***) всередині ***src/main/java***



Частина 2. Створення ієрархії класів

Для зберігання коду цих класів в пакеті проєкту створимо підпакет ***data***. Базовий клас ієрархії ***Shape*** оголосимо абстрактним, він буде містити абстрактний (без реалізації в базовому класі) метод ***draw***, який реалізовуватимемо у похідних класах. Клас можна оголосити абстрактним навіть якщо він не містить жодного абстрактного методу.

В базовому класі ***Shape*** задекларуємо традиційні для геометричних фігур на екрані поля: координати лівого верхнього кута, розміри прямокутника, в якому міститься фігура та її колір. В похідних класах ці параметри обчилюватимемо з розмірів та розташування конкретного типу фігури.

Реалізовуємо потрібні конструктори та методи-аксесори. Орієнтовний код базового класу ***Shape*** :

```
package net.starbasic.minipaint.data;

import java.awt.*;
import java.io.Serializable;

public abstract class Shape implements Serializable {
    // Розташування лівого верхнього кута фігури
```

```
    private int top;
    private int left;
    private int height;// висота у px
    private int width;// ширина у px
    private String color;

    //Конструктор з параметрами (конструктор ініціалізації)
    public Shape(int top, int left, int height, int width,
String color) {
        this.top = top;
        this.left = left;
        this.height = height;
        this.width = width;
        this.color = color;
    }

    public Shape(int top, int left, int height, int width) {
        // делегування попередньому конструктору
        this(top, left, height, width, "black");
    }

    public Shape() {
        // числові поля класу заповнюються нулем автоматично
        this.color = "black";
    }

    public int getTop() {
        return top;
    }

    public void setTop(int top) {
        this.top = top;
    }

    public int getLeft() {
        return left;
    }

    public void setLeft(int left) {
        this.left = left;
    }

    public int getHeight() {
        return height;
    }
```

```
    public void setHeight(int height) {
        this.height = height;
    }

    public int getWidth() {
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public abstract void draw(Graphics2D g);

}
```

В похідних класах реалізовуємо потрібні конструктори (конструктори не успадковуються), метод ***draw***, а також метод ***toString*** з головного базового класу ***Object***. Код похідних класів:

```
package net.starbasic.minipaint.data;

import java.awt.*;

public class Line extends  Shape{

    public Line(int xFrom, int yFrom, int xTo,
                                int yTo, String color) {
        super(yFrom, xFrom, yTo - yFrom, xTo - xFrom, color);
    }

    public Line(int xFrom, int yFrom, int xTo, int yTo) {
        super(yFrom, xFrom, yTo - yFrom, xTo - xFrom);
    }

    public Line() {
     // конструктор за замовчуванням неявно викликає Shape()
    }
```

```
@Override
public void draw(Graphics2D graph) {
    int right = this.getLeft() + this.getWidth();
    int bottom = this.getTop() + this.getHeight();
    switch(this.getColor().toLowerCase()){
        case "red":
            graph.setColor(Color.RED);
            break;
        case "green":
            graph.setColor(Color.GREEN);
            break;
        case "blue":
            graph.setColor(Color.BLUE);
            break;
        case "yellow":
            graph.setColor(Color.YELLOW);
            break;
        case "magenta":
            graph.setColor(Color.MAGENTA);
            break;
        case "cyan":
            graph.setColor(Color.CYAN);
            break;
        case "pink":
            graph.setColor(Color.PINK);
            break;
        case "white":
            graph.setColor(Color.WHITE);
            break;
        default:
            graph.setColor(Color.BLACK);
            break;
    }
    graph.drawLine(this.getLeft(), this.getTop(),
                                        right, bottom);
}

@Override
public String toString() {
    int x2 = this.getLeft() + this.getWidth();
    int y2 = this.getTop() + this.getHeight();
    int x1 = this.getLeft();
    int y1 = this.getTop();
    return String.format("Line from (%4d; %4d) " +
    "to (%4d; %4d), color %s", x1, y1, x2, y2,
                              this.getColor());
```

```
    }
}
package net.starbasic.minipaint.data;

import java.awt.*;

public class Line extends  Shape{
    public Line(int xFrom, int yFrom, int xTo, int yTo,
                                            Color color) {
        super(yFrom, xFrom, yTo - yFrom, xTo - xFrom, color);
    }

    public Line(int xFrom, int yFrom, int xTo, int yTo) {
        super(yFrom, xFrom, yTo - yFrom, xTo - xFrom);
    }

    public Line() {
    // конструктор за замовчуванням неявно викликає Shape()
    }


    @Override
    public void draw(Graphics2D graph) {
        int right = this.getLeft() + this.getWidth();
        int bottom = this.getTop() + this.getHeight();
        graph.setColor(this.getColor());
        graph.drawLine(this.getLeft(), this.getTop(),
                                        right, bottom);
    }

    @Override
    public String toString() {
        int x2 = this.getLeft() + this.getWidth();
        int y2 = this.getTop() + this.getHeight();
        int x1 = this.getLeft();
        int y1 = this.getTop();
        return String
   .format("Line from (%4d; %4d) to (%4d; %4d), color %s",
                               x1, y1, x2, y2,
                               this.getColor().toString());
    }
}
```

```
package net.starbasic.minipaint.data;

import java.awt.*;

public class Rectangle extends Shape{

    public Rectangle(int top, int left, int height
                            int width, String color) {
        super(top, left, height, width, color);
    }

    public Rectangle(int top, int left, int height,
                                            int width) {
        super(top, left, height, width);
    }

    public Rectangle() {
    }

    @Override
    public void draw(Graphics2D graph) {
        switch(this.getColor().toLowerCase()){
            case "red":
                graph.setColor(Color.RED);
                break;
            case "green":
                graph.setColor(Color.GREEN);
                break;
            case "blue":
                graph.setColor(Color.BLUE);
                break;
            case "yellow":
                graph.setColor(Color.YELLOW);
                break;
            case "magenta":
                graph.setColor(Color.MAGENTA);
                break;
            case "cyan":
                graph.setColor(Color.CYAN);
                break;
            case "pink":
                graph.setColor(Color.PINK);
                break;
            case "white":
                graph.setColor(Color.WHITE);
                break;
```

```
            default:
                graph.setColor(Color.BLACK);
                break;
        }
        graph.fillRect(this.getLeft(), this.getTop(),
                   this.getWidth(),this.getHeight());
    }

    @Override
    public String toString() {
    return String.format("Rectangle %3d x %3d at (%3d; %3d)",
                this.getWidth(), this.getHeight(),
                this.getTop(), this.getLeft())
                + this.getColor();
    }
}
```

```
package net.starbasic.minipaint.data;

import java.awt.*;

public class Circle extends Shape{

    public Circle(int x, int y, int R, String color) {
        super(y-R, x-R, 2*R, 2*R, color);
    }

    public Circle(int x, int y, int R) {
        super(y-R, x-R, 2*R, 2*R);
    }

    public Circle() {
    }

    @Override
    public String toString() {
        int d = this.getWidth();
        double r = ((double) d) / 2;
        double x = this.getLeft() + r;
        double y = this.getTop() + r;
        return String.format("Circle (%7.2f, %7.2f), ",x,y) +
              String.format("radius %7.2f,",r) +
                       this.getColor().toString();
    }
```

```
    @Override
    public void draw(Graphics2D graph) {
        switch(this.getColor().toLowerCase()){
            case "red":
                graph.setColor(Color.RED);
                break;
            case "green":
                graph.setColor(Color.GREEN);
                break;
            case "blue":
                graph.setColor(Color.BLUE);
                break;
            case "yellow":
                graph.setColor(Color.YELLOW);
                break;
            case "magenta":
                graph.setColor(Color.MAGENTA);
                break;
            case "cyan":
                graph.setColor(Color.CYAN);
                break;
            case "pink":
                graph.setColor(Color.PINK);
                break;
            case "white":
                graph.setColor(Color.WHITE);
                break;
            default:
                graph.setColor(Color.BLACK);
                break;
        }
        int d = this.getWidth();
        int x = this.getLeft() ;
        int y = this.getTop();
        graph.fillOval(x, y, d, d);
    }
}
```

Частина 3. Робота з колекцією об'єктів.

Для зберігання набору геометричних фігур створимо окремий клас, в якому інкапсулюємо список (наприклад, ArrayList) посилань на об'єкти класу Shape (точніше його нащадків, з врахуванням поліморфізму):

```
package net.starbasic.minipaint.data;

import java.awt.*;
import java.util.ArrayList;

public class ShapesCollection {
    private final ArrayList<Shape> shapes;

    public ShapesCollection() {
        shapes = new ArrayList<>();
    }

    public void append(Shape shape){
        shapes.add(shape);

    }

    public void drawAll(Graphics2D graphics){
        for (Shape shape: this.shapes){
            shape.draw(graphics);
        }
    }

    public static ShapesCollection fromFile(String path){
        ShapesCollection collection = new ShapesCollection();

        //TODO
        return  collection;
    }


    public void saveToFile(String path){
        // TODO
    }
}
```

Методи з коментарем TODO, реалізуємо згодом.

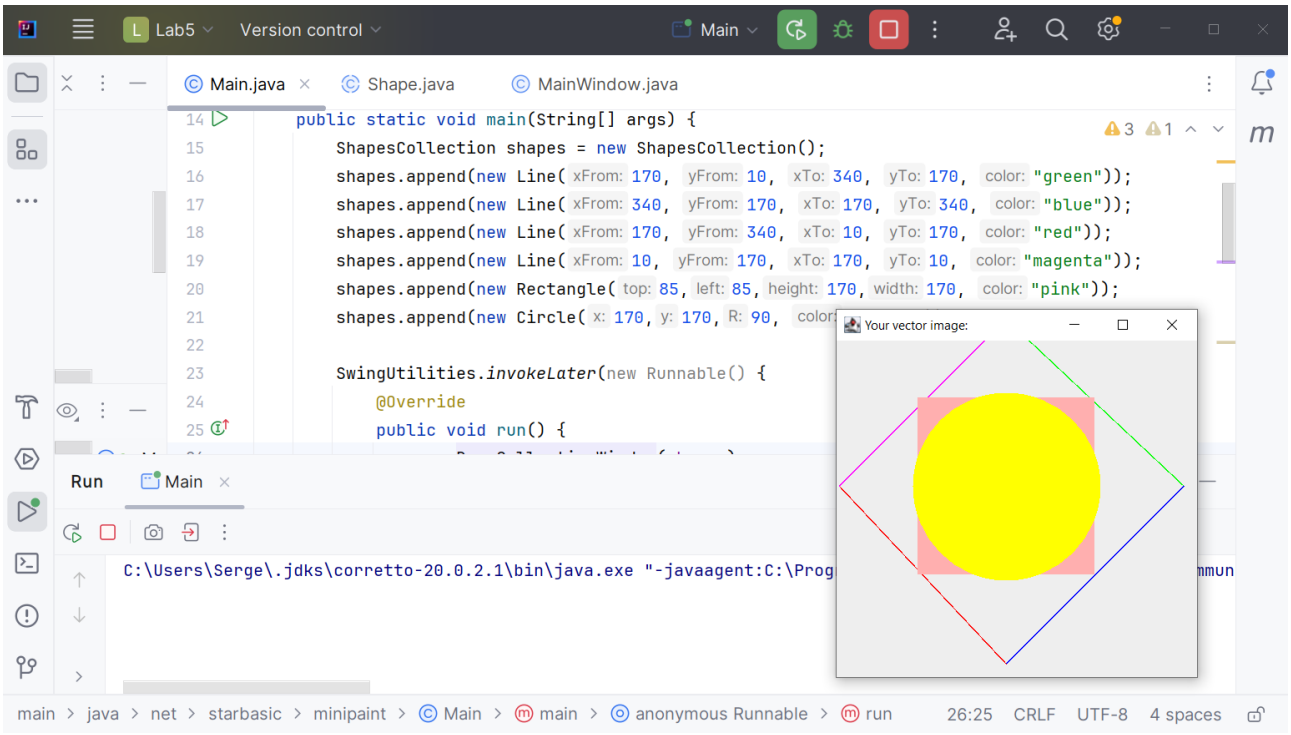
Для попередньої демострації можливостей графічної візуалізації стореного набору класів можемо скористатися таким кодом:

```
package net.starbasic.minipaint;

import net.starbasic.minipaint.data.ShapesCollection;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Line2D;

public class DrawCollectionWindow extends JFrame {

    private ShapesCollection _shapesCollection;
    public DrawCollectionWindow(
              ShapesCollection shapesCollection) {
        super("Your vector image:");
        setSize(360, 360);
        this._shapesCollection = shapesCollection;
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    void drawShapes(Graphics graphics) {
        Graphics2D graph2d = (Graphics2D) graphics;
        this._shapesCollection.drawAll(graph2d);
    }

    public void paint(Graphics g) {
        super.paint(g);
        drawShapes(g);
    }
}
```

Клас, код якого подано вище, забезпечує створення програмним шляхом нового вікна типу JFrame та малювання на ньому усіх фігур з переданої в конструкторі колекції фігур. В подальшому цей код можна буде використати як один з елементів функціоналу основної частини проєкту. Для демонстрації відображення деякої колекції фігур з допомогою екземпляра класу ***DrawCollectionWindow*** можемо скористатися наступним кодом:

```
package net.starbasic.minipaint;
import net.starbasic.minipaint.data.*;
import javax.swing.*;
import java.awt.*;
```

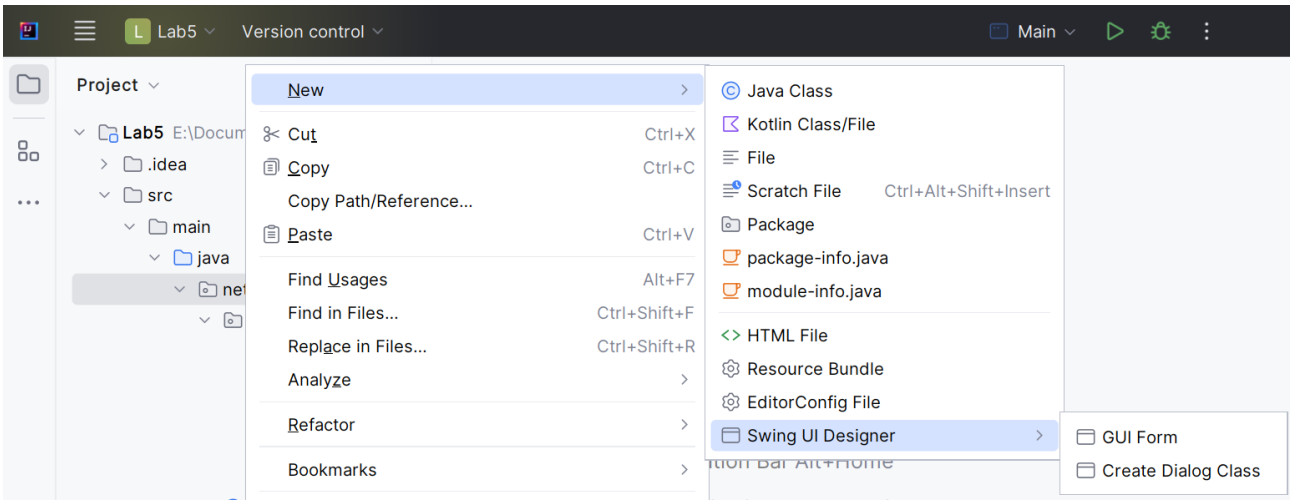
```
public class Main {
  public static void main(String[] args) {
      ShapesCollection shapes = new ShapesCollection();
      shapes.append(new Line(170, 10, 340, 170, "green"));
      shapes.append(new Line(340, 170, 170, 340, "blue"));
      shapes.append(new Line(170, 340, 10, 170, "red"));
      shapes.append(new Line(10, 170, 170, 10, "magenta"));
      shapes.append(new Rectangle(85,85,170,170, "pink"));
      shapes.append(new Circle(170,170,90, "yellow"));

          SwingUtilities.invokeLater(new Runnable() {
              @Override
              public void run() {
                  new DrawCollectionWindow(shapes)
                                  .setVisible(true);
              }
          });
      }
}
```



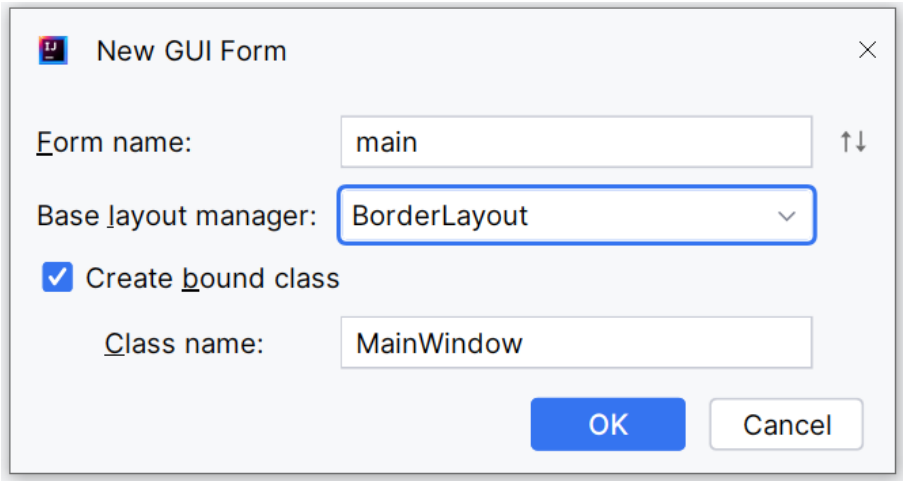
Частина 4. Конструкція вікона для опрацювання списку фігур.

Створимо вікно для редагування списку геометричних фігур відповідно до варіанта. Як ми пересвідчилися на прикладі вище, Java Swing дозволяє створювати вікна та їх елементи безпосередньо з коду програми.

IntelliJ IDEA надає розробникові можливість використовувати для проектування віконного інтерфейсу вбудований візуальний редактор. Для його використання додамо до проєкту ***GIU Form*** з пункту ***Swing UI*** контекстного меню відповідного пакета у вікні проєкту

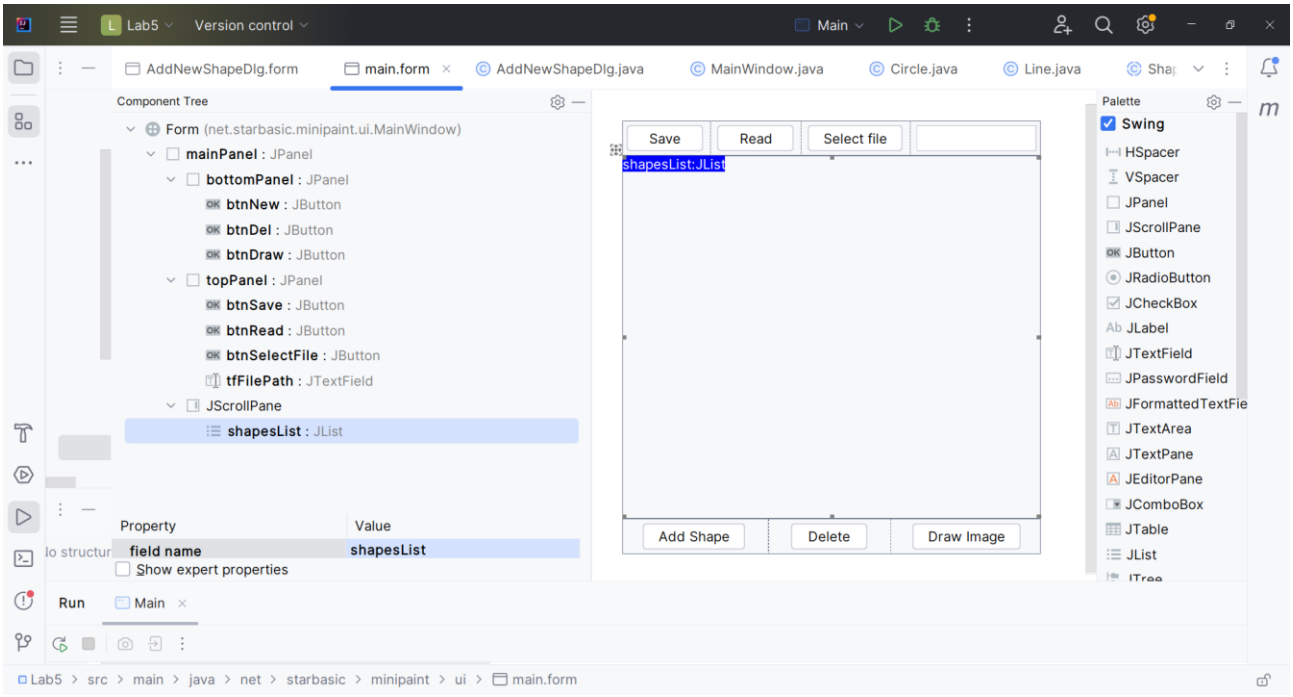


Створимо спочатку пакет ui в кореневому пакеті нашого проєкту, та додамо до нього ***GIU Form*** для головного вікна нашої програми:



Відмітимо прапорець ***Create bound class*** для того, щоб відразу створити клас в якого можна програмувати роботу нашого віконного інтерфейсу.

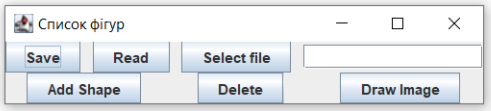
Менеджер компоновки можна залишити за замовчуванням (***Border Layout***). У його центрі помістимо ***JScrollPane*** з ***JList*** для відображення списку фігур, – у верхній частині ***JPanel*** для кнопок, що керуватимуть читанням-записом списку фігур у файлі, у нижній – ***JPanel*** для кнопок, що керуватимуть додаванням-вилученням фігур у списку. Додаємо до форми потрібні елементи керування. Розташування контролів та їх назви показано на знімку нижче:



IntelliJ IDEA створює окремо розмітку елементів вікна (в форматі xml) та окремий клас (в нашому випадку ***MainWindow***). Об'єднати їх в робочий функціонал можна різними способами, ми скористаємося спадкуванням від базового класу swing-фрейма ***JFrame*** у класі ***MainWindow*** створенням його екземпляра та відображенням його в в методі ***main*** головного класу, подібно до прикладу з класом ***DrawCollectionWindow*** вище. При цьому слід врахувати, що усі наші візуальні компоненти розташовані на окремій панелі (див. зображення вище) та додати її в фрейм у конструкторі класу ***MainWindow***, наприклад:

```
public MainWindow(ShapesCollection shapes)
                            throws HeadlessException {
        // Конструктор класу JFrame, з текстом заголовку
        super("Список фігур");
        // всановлення панелі з компонентами на фреймі
        this.setContentPane(mainPanel);
        // розміщення фрейму в центрі екрану
        this.setLocationRelativeTo(null);
        // Завершення усієї програми разом із
        // закриттям цього вікна
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        listModel.addAll((Collection<? extends Shape>)
        // встановлення розмірів фрейма відповідно до вмісту
        this.pack();
    }
```

```
    private JButton btnSelectFile;
    1 usage
    private JTextField tfFilePath;
    1 usage
    private JPanel bottomPanel;
    1 usage
    private JPanel topPanel;
    2 usages
    private JPanel mainPanel;
    1 usage
    public MainWindow() throws HeadlessException {
        // Конструктор базового класу JFrame,
        // що приймає текст заголовку
        super( title: "Список фігур");
        // всановлення панелі з компонентами на фрейм
        this.setContentPane(mainPanel);
        // розміщення фрейму в центрі екрану
        this.setLocationRelativeTo(null);
        // втановлення розмірів фрейма відповідно до вмісту
        this.pack();

    }
}
```



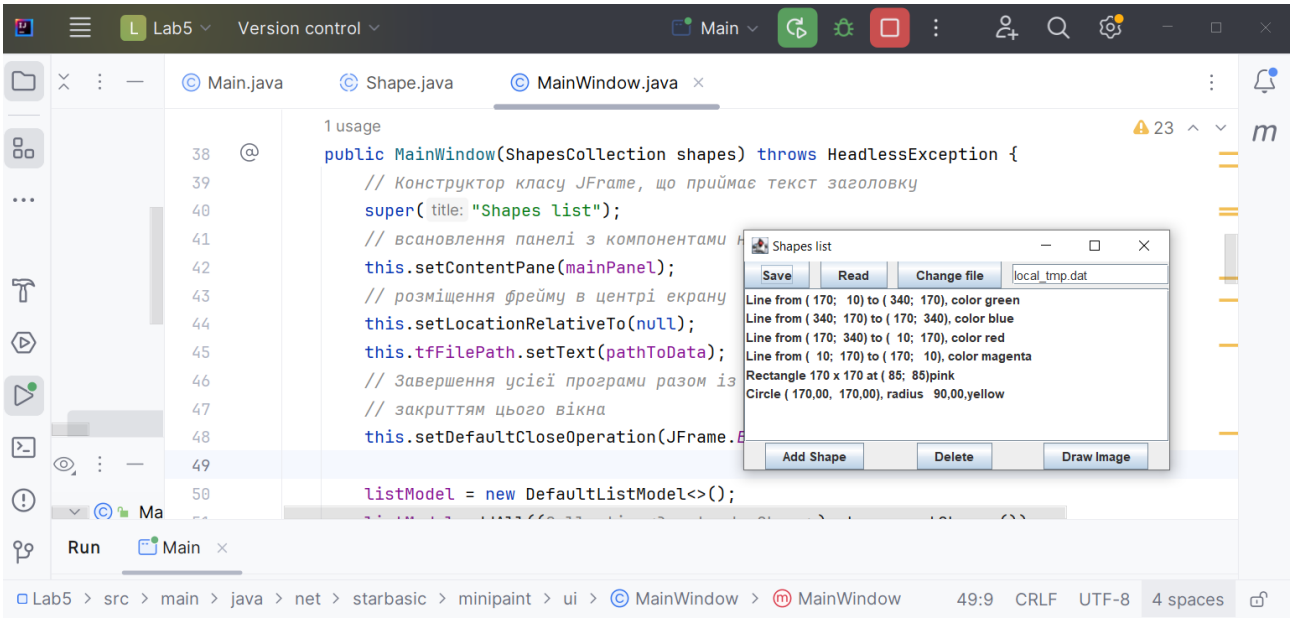
Main

Наразі список порожній тому відображаються лише елементи керування. Можна задати вікну фіксовані розміри, наприклад мінімальну висоту, тоді побачимо порожній список у вікні.

Для наповнення списку можемо скористатися колекцією ***ShapesCollection***, що була створена в головному методі консольного варіанту. При цьому слід врахувати, що:

- ✓ цю колекцію потрібно передати в конструктор класу ***MainWindow***;
- ✓ компонент ***JList*** реалізовано за патерном MVC, тому йому слід сворити модеть на основі переданої колекції;
- ✓ в клас ***ShapesCollection*** потрібно додати методи для обміну колекцією об’єктів з моделлю даних компонента ***JList*** (наприклад, гетер до поля ***shapes*** та конструктор, що приймає список елементів типу ***Shape***);
- ✓ у списку відображатимуться текст з інформацією про об’єкти, тому усім нащадкам класу ***Shape*** потрібно перевизначити «достатньо інформативний» метод ***toString()*** дефолтного, успадкованого від ***Object***.

Створений раніше клас ***DrawCollectionWindow*** з фреймом для графічного відображення списку фігур пернесемо в пекет ui, внісши відповідні зміни в код, де це портібно.



Частина 5. Опрацювання подій елементів керування в Java Swing.

Для обробки подій в Java Swing використовується механізм так званих «***слухачів***» ***Listener***. Їх оголошено в бібліотечних класах візуальних компонентів за допомогою функціональних інтерфейсів, методи яких викликаються при настанні в компоненті відповідної події. Щоб реалізувати опрацювання цієї події потрібно імплементувати відповідний інтерфейс та зареєструвати свою імплементацію слухачем для обраного елемента вікна.

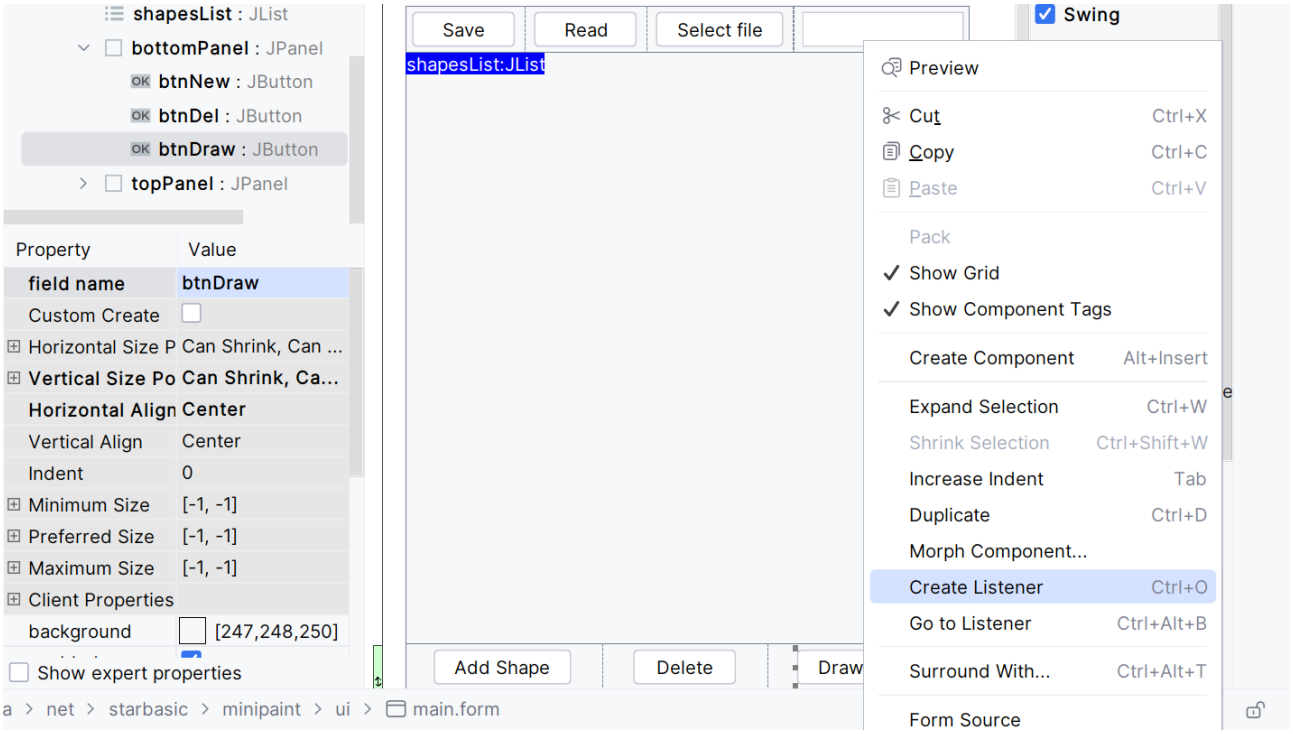
Імплементацію ***Listener***-а можна оформити у вигляді окремого класу. Практикують також створення цілих класів-диспетчерів з імплементаціями ***Listener***-ів для всіх компонентів вікна. Але найпростіший спосіб, яким ми і скористаємося – реалізація слухача у вигляді анонімного класу безпосередньо в місці реєстрації обробника події. Останній спосіб, починаючи з Java 1.8, дозволяє також використовувати ***Lambda-вирази***.

Реалізуємо обробник для кнопки «Draw Image», який буде запускати додатковий фрейм з графічним зображенням на основі класу ***DrawCollectionWindow***. Для створення обробника в конструктор головного фрейму потрібно додати фрагмент коду:

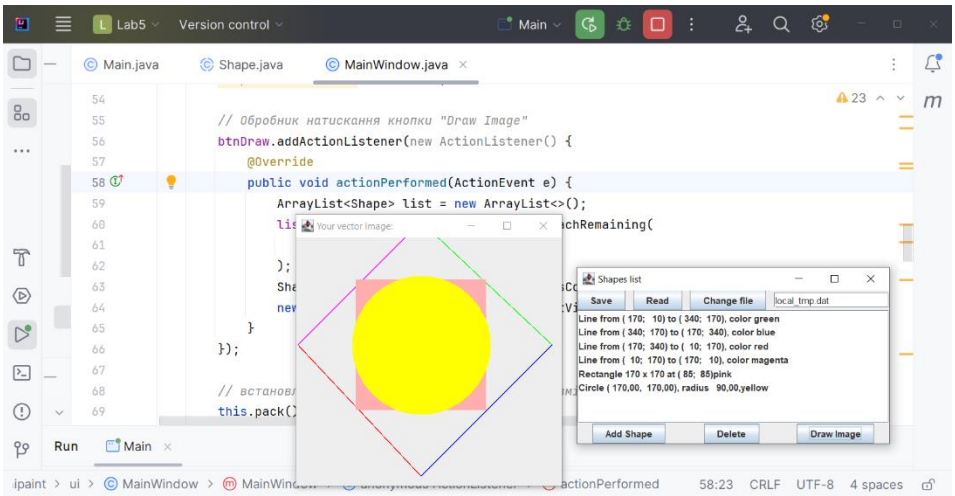
```
// Обробник натискання кнопки "Draw Image"
btnDraw.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    // Зчитуємо актуальний список фігур з головного вікна
    // та передаємо у вікно для малювання
    ArrayList<Shape> list = new ArrayList<>();
    listModel.elements().asIterator().forEachRemaining(
                        s->list.add(s)
    );
    ShapesCollection allShapes = new ShapesCollection(list);
```

```
        new DrawCollectionWindow(allShapes).setVisible(true);
    }
});
```

IntelliJ IDEA надає розробникові зручний спосіб створення ***`Listener`***-а за допомогою контекстного меню. Для цього слід натиснути праву клавішу миші на відповідному компоненті та обрати ***`Create Listener`***.



Далі потрібно обрати відповідний інтерфейс для реалізації, у випадку кнопок це ***`ActionListener`***. На наступному кроці – вказати метод інтерфейсу для реалізації. Більшість слухачів, в тому числі ***`ActionListener`***, є функціональними інтерфейсами, тобто містять лише один метод. Тому на цьому кроці достатньо погодитися з вибором IDE.

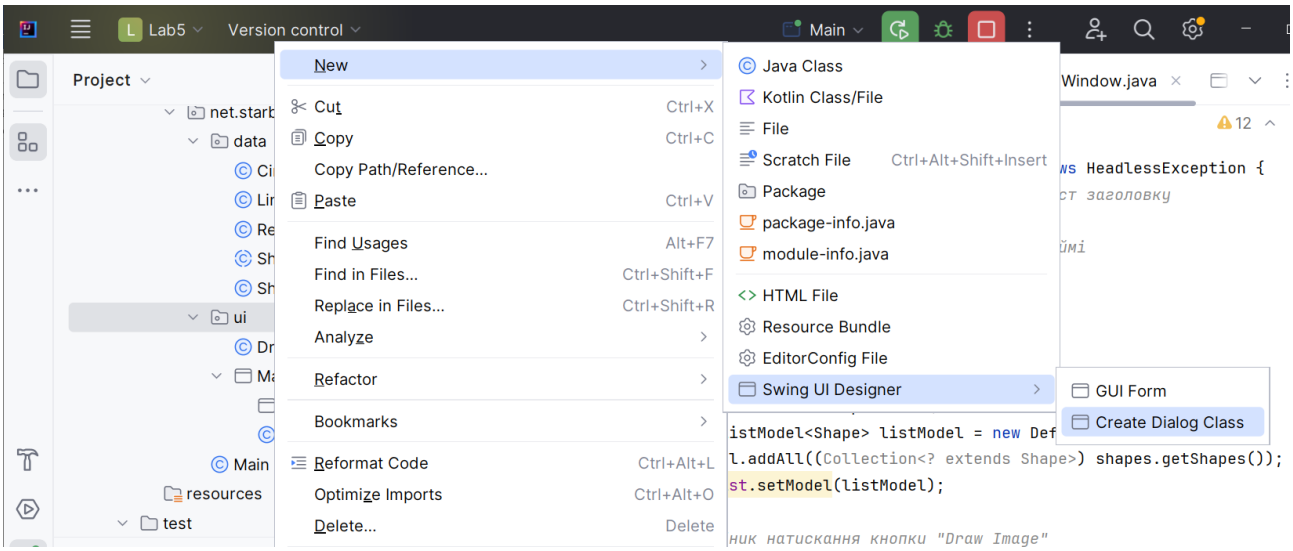


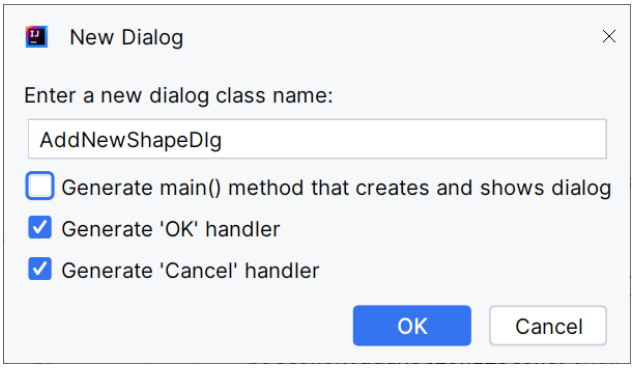
Результат роботи створеного ***`ActionListener`***-а.

Код решти обробників подано в лістингу у кінці прикладу.

Частина 6. Створення діалогового вікна для додавання нового елемента в список.

Для додавання нового елемента в список скористаємося діалоговим вікном на основі класу ***javax.swing.JDialog***. Для цього до пакету ui додамо новий ***Dialog Class*** з розділу ***Swing UI Designer*** (див. малюнок нижче).





Задамо класові діалогового вікна назву, наприклад, ***AddNewShapeDlg***. Тут, на відміну від створення ***JFrame***, IDE пропонує нам згенерувати код для методу ***main*** та обробників натискання кнопок. Приймемо пропозицію, залишивши прапорці лише для обробників. Метод ***main*** нам не потрібен, бо відображати діалог ми будемо з обробника відповідної книпки на головному вікні.

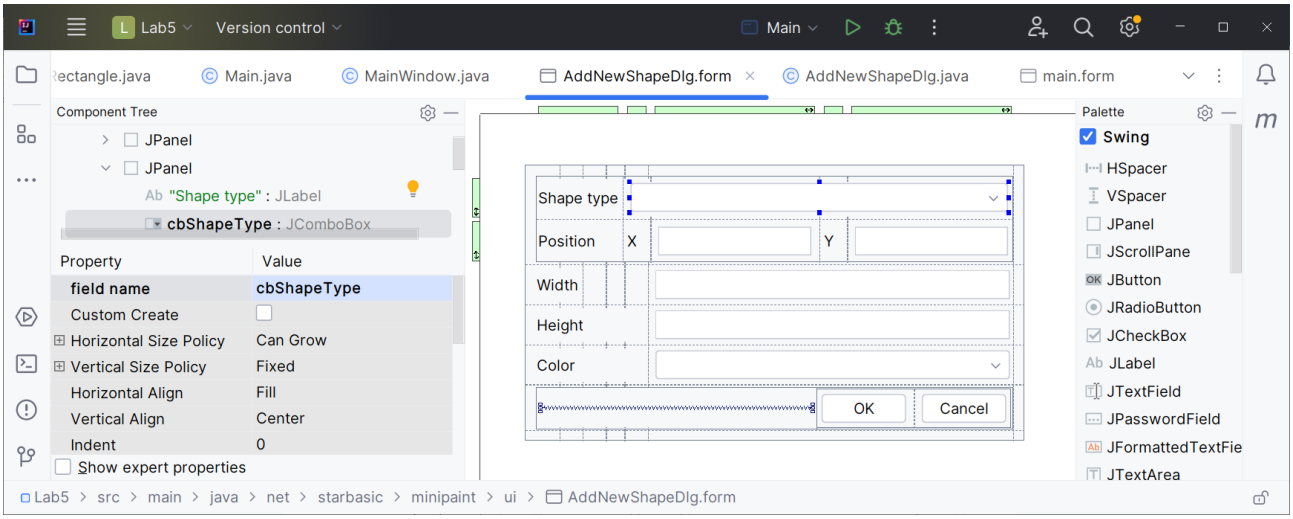
Вікно ***JDialog***, на відміну ***JFrame***, може працювати в режимі модального діалогу, що дозволяє призупинити виконання всіх інших процесів додатку доки в діалог не буде внесена потрібна інформація та підтверджено її ввід.

У візуальному редакторі додаємо на діалог необхіді компоненти. В нашому випадку це:

- два JComboBox – для вибору типу фігури та її кольору;
- чотири JTextField – для вводу координат розташування та розміру фігури;
- підписи JLabel для позначення відповідних полів.

Зрозуміло, що видимість та підписи деяких компонентів залежатимуть від типу фігури.

Орієнтовний вигляд діалогового вікна.



Орієнтовний код класу діалогового вікна:

```
package net.starbasic.minipaint.ui;

import net.starbasic.minipaint.data.Circle;
import net.starbasic.minipaint.data.Line;
import net.starbasic.minipaint.data.Rectangle;
import net.starbasic.minipaint.data.Shape;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class AddNewShapeDlg extends JDialog {
  private JPanel contentPane;
  private JButton buttonOK;
  private JButton buttonCancel;
  private JComboBox cbShapeType;
  private JTextField tfPosX;
  private JTextField tfPosY;
  private JTextField tfWidth;
  private JTextField tfHeight;
  private JComboBox cbShapeColor;
  private JLabel labelWidth;
  private JLabel labelHeight;
  private JLabel labelPosition;



  private String colors[] = {"white", "red", "green", "blue",
          "yellow", "black", "magenta", "cyan", "pink"};
```

```
  private  String shapeTypes[] = {"Rectangle", "Circle",
"Line"};
  public AddNewShapeDlg(Shape results[]) {
    setContentPane(contentPane);
    setModal(true);
    getRootPane().setDefaultButton(buttonOK);
    setLocationRelativeTo(null);
    pack();
    results[0] = new Rectangle();

    // Опрацювання вибору типу фігури зі списку
    cbShapeType.setModel(new
DefaultComboBoxModel<String>(shapeTypes));
    cbShapeType.addItemListener(e -> {
      // метод для обробки викликаємо з коду класу
      changeShapeType(results);

    });

    // Вибір кольору фігури зі списку
    cbShapeColor.setModel(new
DefaultComboBoxModel<String>(colors));

    buttonOK.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        onOK(results);
      }
    });

    buttonCancel.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        onCancel();
      }
    });

    // call onCancel() when cross is clicked
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        onCancel();
      }
    });

    // call onCancel() on ESCAPE
    contentPane.registerKeyboardAction(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
        onCancel();
      }
    }, KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
  }

  private void changeShapeType(Shape results[]) {
    switch (cbShapeType.getSelectedItem().toString()){
      case "Circle":
        results[0] = new Circle();
        labelPosition.setText("Center");
        labelHeight.setVisible(false);
        labelWidth.setText("Radius");
        tfHeight.setVisible(false);
        break;
      case "Line":
        results[0] = new Line();
        labelPosition.setText("Start");
        labelHeight.setVisible(true);
        labelHeight.setText("EndY");
        labelWidth.setText("EndX");
        tfHeight.setVisible(true);
        break;
      case "Rectangle":
        results[0] = new Rectangle();
        labelPosition.setText("Position");
        labelHeight.setVisible(true);
        labelHeight.setText("Height");
        labelWidth.setText("Width");
        tfHeight.setVisible(true);
        break;
    }

  }

  private void onOK(Shape results[]) {
    int x = Integer.parseInt(tfPosX.getText());
    int y = Integer.parseInt(tfPosY.getText());

    switch (cbShapeType.getSelectedItem().toString()){
      case "Circle":
        int R = Integer.parseInt(tfWidth.getText());
        results[0].setLeft(x - R);
        results[0].setTop(y - R);
        results[0].setHeight(2 * R);
        results[0].setWidth(2 * R);
```

```
        break;
      case "Line":
        int x2 = Integer.parseInt(tfWidth.getText());
        int y2 = Integer.parseInt(tfHeight.getText());
        results[0].setLeft(x);
        results[0].setTop(y);
        results[0].setHeight(y2 - y);
        results[0].setWidth(x2 - x);
        break;
      case "Rectangle":
        int width = Integer.parseInt(tfWidth.getText());
        int height = Integer.parseInt(tfHeight.getText());
        results[0].setLeft(x);
        results[0].setTop(y);
        results[0].setHeight(height);
        results[0].setWidth(width);
        break;
    }

results[0].setColor(cbShapeColor.getSelectedItem().toString()
);
    dispose();
  }

  private void onCancel() {
    dispose();
  }


}
```

Частина 7. Реалізація зберігання списку фігур у файлі.

Реалізовуємо методи, які ми раніше позначили коментарем ***//TODO*** в класі ***ShapesCollection***. Код класу тепер виглядатиме так:

```
public class ShapesCollection {
  private final ArrayList<Shape> shapes;

  public ShapesCollection() {
    shapes = new ArrayList<>();
  }

  public ShapesCollection(ArrayList<Shape> shapes) {
    this.shapes = shapes;
  }
```

```
public void append(Shape shape){
  shapes.add(shape);
}

public void drawAll(Graphics2D graphics){
  for (Shape shape: this.shapes){
    shape.draw(graphics);
  }
}

public static ShapesCollection fromFile(String path){
  ArrayList<Shape> shapesList = new ArrayList<>();
  if ((new File(path)).exists()) {
    try (FileInputStream fileInputStream =
           new FileInputStream(path)) {
      try (ObjectInputStream objectInputStream =
             new ObjectInputStream(fileInputStream)) {
        shapesList = (ArrayList<Shape>) objectInputStream
            .readObject();
      } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
      }
    } catch (FileNotFoundException e) {
      throw new RuntimeException(e);
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
  }
  return  new ShapesCollection(shapesList);
}

public void saveToFile(String path){
  try(FileOutputStream fileOutputStream =
        new FileOutputStream(path)){
    try(ObjectOutputStream objectOutputStream =
          new ObjectOutputStream(fileOutputStream)){
      objectOutputStream.writeObject(this.shapes);
    }
  } catch (FileNotFoundException e) {
    throw new RuntimeException(e);
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
}
```

```
  public ArrayList<Shape> getShapes() {
    return shapes;
  }
}
```

Додаємо обробники для решти кнопок головного вікна зі списком. Орієнтовний код головного вікна програми:

```
package net.starbasic.minipaint.ui;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import net.starbasic.minipaint.data.Shape;
import net.starbasic.minipaint.data.ShapesCollection;
import javax.swing.*;
import java.awt.HeadlessException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

public class MainWindow extends JFrame {
  private JList shapesList;
  private JButton btnNew;
  private JButton btnDraw;
  private JButton btnDel;
  private JButton btnSave;
  private JButton btnRead;
  private JButton btnSelectFile;
  private JTextField tfFilePath;
  private JPanel bottomPanel;
  private JPanel topPanel;
  private JPanel mainPanel;
  // шлях до файла в якому зберігається колекція
  String pathToData = "local_tmp.dat";
  // модель для списку JList
  DefaultListModel<Shape> listModel;
  public MainWindow() throws HeadlessException {
    // Конструктор класу JFrame, що приймає текст заголовку
```

```
    super("Shapes list");
    // всановлення панелі з компонентами на фреймі
    this.setContentPane(mainPanel);
    // розміщення фрейму в центрі екрану
    this.setLocationRelativeTo(null);
    this.tfFilePath.setText(pathToData);
    // Завершення усієї програми разом із
    // закриттям цього вікна
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ShapesCollection shapes =
ShapesCollection.fromFile(pathToData);

    listModel = new DefaultListModel<>();
    listModel.addAll((Collection<? extends Shape>)
shapes.getShapes());
    shapesList.setModel(listModel);

    // Обробник натискання кнопки "Draw Image"
    btnDraw.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        ArrayList<Shape> list = new ArrayList<>();
        listModel.elements().asIterator().forEachRemaining(
            s->list.add(s)
        );
        ShapesCollection allShapes =
                                new ShapesCollection(list);
        new DrawCollectionWindow(allShapes).setVisible(true);
      }
    });
    // встановлення розмірів фрейма відповідно до вмісту
    this.pack();

    btnNew.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        // На момент запуску діалогу тип нащадка Shape
        // невідомий, використаємо масив для отримання
        // посилання на об'єкт сворений в іншому потоці
        Shape result[] = new Shape[1];
        AddNewShapeDlg newShapeDialog =
                            new AddNewShapeDlg(result);
        newShapeDialog.setModal(true);
        newShapeDialog.setVisible(true);
        if(result[0] != null) {
          listModel.addElement(result[0]);
```

```
    }
  }
});

btnDel.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    listModel.remove(shapesList.getSelectedIndex());
  }
});
btnSelectFile.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    JFileChooser jFileChooser = new JFileChooser();
    int result = jFileChooser.showSaveDialog(null);
    File file;
    if(result==JFileChooser.APPROVE_OPTION){
      file = jFileChooser.getSelectedFile();
      pathToData = file.getPath();
      tfFilePath.setText(file.getName());
    }
  }
});
btnSave.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    ArrayList<Shape> list = new ArrayList<>();
    listModel.elements().asIterator().forEachRemaining(
        s->list.add(s)
    );
    ShapesCollection allShapes =
                          new ShapesCollection(list);
    allShapes.saveToFile(pathToData);
  }
});
btnRead.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    JFileChooser jFileChooser = new JFileChooser();
    int result = jFileChooser.showOpenDialog(null);
    File file;
    if(result==JFileChooser.APPROVE_OPTION){
      file = jFileChooser.getSelectedFile();
      pathToData = file.getPath();
      tfFilePath.setText(file.getName());
    }
```

```
            ShapesCollection newCollection =
                    ShapesCollection.fromFile(pathToData);
            listModel.clear();
            listModel.addAll(
        (Collection<? extends Shape>) newCollection.getShapes());
          }
        });
    }
}
```

Після того, як програма отримала можливість зберігати список фігур у файлі можемо прибрати код, що створював тимчасову тестову колекцію у методі ***main***.

```
package net.starbasic.minipaint;

import net.starbasic.minipaint.data.Circle;
import net.starbasic.minipaint.data.Line;
import net.starbasic.minipaint.data.Rectangle;
import net.starbasic.minipaint.data.ShapesCollection;
import net.starbasic.minipaint.ui.DrawCollectionWindow;
import net.starbasic.minipaint.ui.MainWindow;

import javax.swing.*;
import java.awt.*;

public class Main {
    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new MainWindow().setVisible(true);
            }
        });
    }
}
```

*** Додаткове завдання: замінити подану у прикладі бінарну серіалізацію-десеріалізацію колекції об'єктів у файл на збереження даниних для роботи програми у форматJSON.*