

## Лабораторна робота №1

**Завдання:** Створити малюнок за варіантом користуючись графічними примітивами бібліотеки JavaFX.

### Методичні вказівки

Для початку потрібно створити клас, що спадкується від класу *Application* та реалізує абстрактний метод класу предка.

```
@Override
public void start(Stage primaryStage){
}
```

Також клас повинен мати метод *main* наступного та містити ньому виклик успадкованого (або перевизначеного) методу *launch*. Наприклад:

```
public static void main(String[] args) {
    launch(args);
}
```

Далі треба створити контейнер, в якому будуть знаходитись усі наші фігури. Для цього скористаємось класом *Group*.

```
Group root = new Group();
```

Посилання на цей об'єкт необхідно передати в конструктор сцени, яка буде візуалізована у вікні нашої програми.

```
Scene scene = new Scene(root, 300, 250);
```

Цим конструктором ми вказали розміри нашої сцени (300x250) та посилання на кореневий елемент сцени (*root*).

```
primaryStage.setScene(scene);
primaryStage.show();
```

Додамо імпорт використаних класів та отримаємо нашу першу *JavaFX* програму.

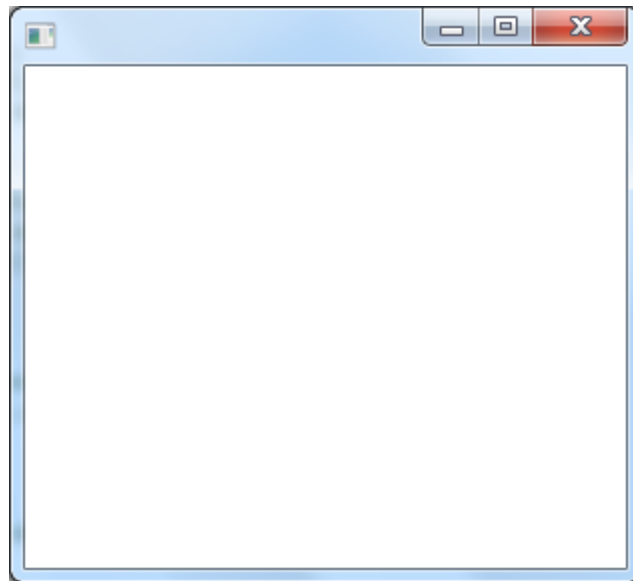
```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
public class MyJavaFXApplication extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        //////////////////////////////////////
        //елементи сцени будемо вставляти сюди
        //////////////////////////////////////
    }
}
```

```

        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```



Отримали вікно з порожньою сценою з розмірами (300x250).

Для використання графічних примітивів їх необхідно імпортувати:

```
import javafx.scene.shape.*;
```

Створимо прямокутник і помістимо його в кореневий контейнер.

```

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;

import javafx.scene.shape.*;

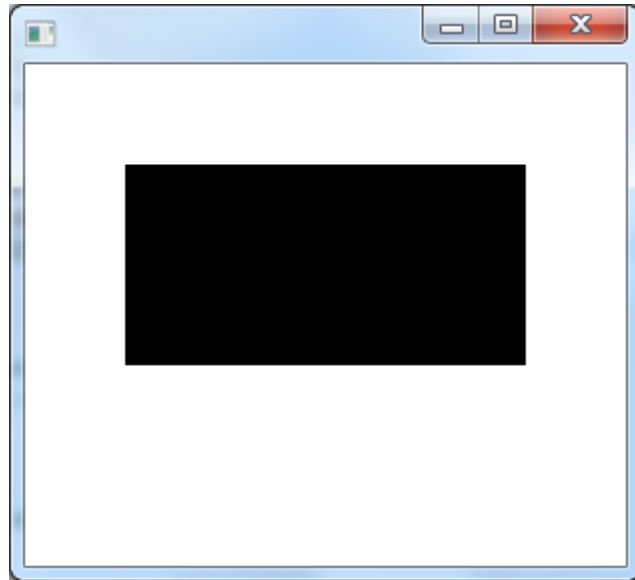
public class MyJavaFXApplication extends Application {
    public static void main(String[] args) {
        Launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        ///////////////////////////////////////////////////
        //елементи сцени будемо вставляти сюди

        Rectangle r = new Rectangle(50, 50, 200, 100); //створення фігури
        root.getChildren().add(r); //додавання фігури до кореневого контейнера

        ///////////////////////////////////////////////////
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```



Інші геометричні примітиви додаються до сцени аналогічно – їх треба створити та додати до кореневого контейнера, який відобразиться у сцені.

Змінемо колір заповнення за допомогою метода фігури *setFill(Paint value)*.

Ми будемо змінювати колір за допомогою класу *Color* (додамо імпорт)

```
import javafx.scene.paint.Color;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.shape.*;

import javafx.scene.paint.Color;

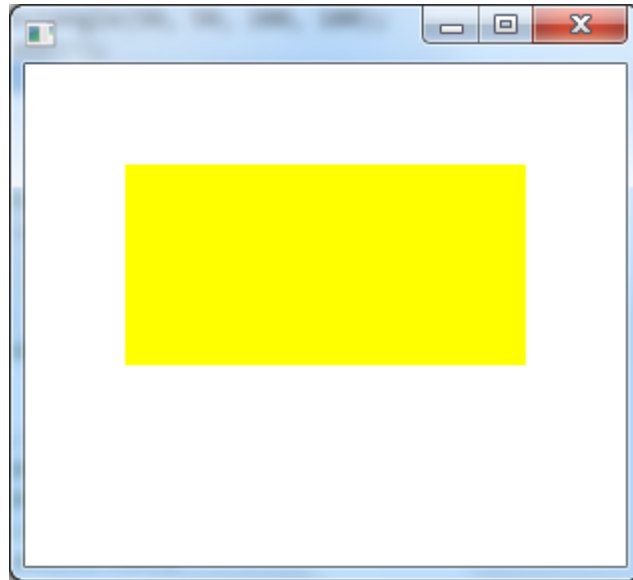
public class MyJavaFXApplication extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        //////////////////////////////////////
        //елементи сцени будемо вставляти сюди

        Rectangle r = new Rectangle(50, 50, 200, 100); //створення фігури
        root.getChildren().add(r); //додавання фігури до кореневого контейнера

        r.setFill(Color.rgb(255, 255, 0)); //Встановимо жовтий колір за допомогою
        кольорової схеми RGB

        //////////////////////////////////////
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



Зазначимо, що замість виклику метода *Color.rgb(255, 255, 0)* можна було скористатися константою *Color.YELLOW*. Кількість наперед визначених констант кольору рахується десятками і при бажанні можете випробувати кожну з них.

Сама сцена також має метод *setFill*, що дозволяє встановити фон для усієї сцени. Цього разу використаємо константи.

```
import javafx.scene.paint.Color;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.shape.*;
import javafx.scene.paint.Color;

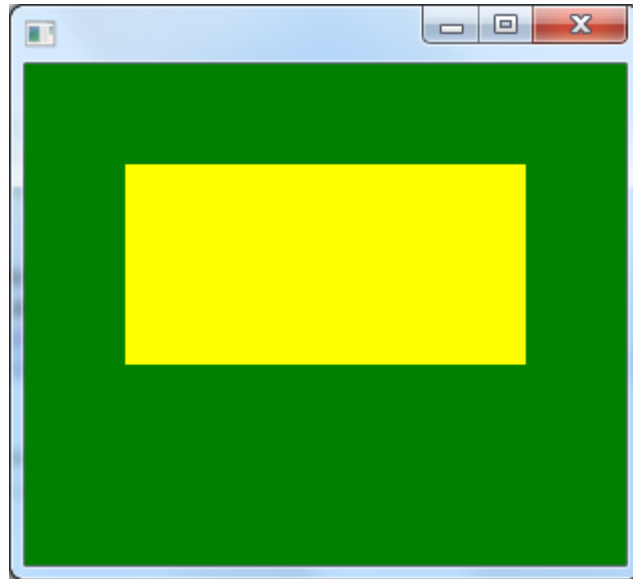
public class MyJavaFXApplication extends Application {
    public static void main(String[] args) {
        Launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        //////////////////////////////////////
        //елементи сцени будемо вставляти сюди

        Rectangle r = new Rectangle(50, 50, 200, 100); //створення фігури
        root.getChildren().add(r); //додавання фігури до кореневого контейнера
        r.setFill(Color.YELLOW); //Встановимо жовтий колір за допомогою константи

        scene.setFill(Color.GREEN); //Встановимо зелений фон за допомогою константи

        //////////////////////////////////////
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



Нагадаємо, що кореневий контейнер – це колекція і до нього можна додавати декілька фігур. Відобразатися вони будуть в порядку додавання до контейнера, тобто новіші будуть на передньому плані, а старіші – на задньому. Для прикладу додамо червоний відрізок.

```
import javafx.scene.paint.Color;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.shape.*;
import javafx.scene.paint.Color;

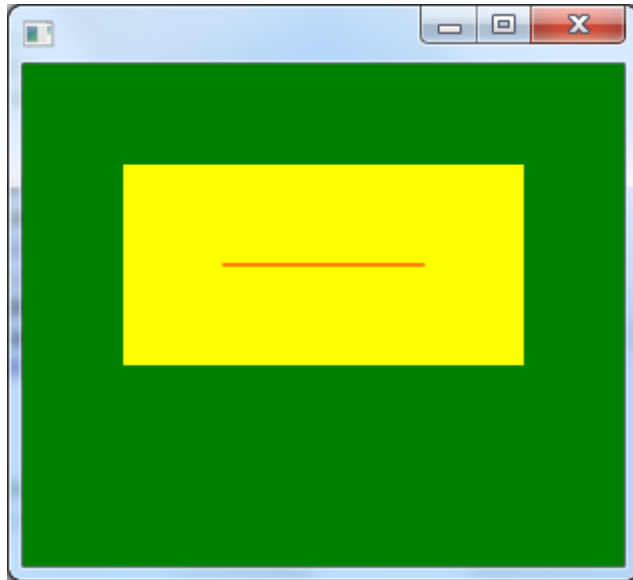
public class MyJavaFXApplication extends Application {
    public static void main(String[] args) {
        Launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        ////////////////////////////////////////
        //елементи сцени будемо вставляти сюди

        Rectangle r = new Rectangle(50, 50, 200, 100); //створення фігури
        root.getChildren().add(r); //додавання фігури до кореневого контейнера
        r.setFill(Color.YELLOW); //Встановимо жовтий колір за допомогою константи
        scene.setFill(Color.GREEN); //Встановимо зелений фон за допомогою константи

        Line l = new Line(100, 100, 200, 100); //
        root.getChildren().add(l);
        l.setStroke(Color.RED);

        ////////////////////////////////////////
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



Відмітимо, що для зміни кольору відрізка ми використали метод *setStroke* замість *setFill*. Річ у тому, що відрізок не має площі яку можна було б заповнити кольором, тому для надання йому кольору використовується колір контуру (за замовчанням – чорний).

Тепер розглянемо створення окремих примітивів:

### **Прямий відрізок**

```
Line line = new Line(0.0f, 0.0f, 100.0f, 100.0f);
```

або

```
Line line = new Line();  
//Встановимо початкову точку  
line.setStartX(0.0f);  
line.setStartY(0.0f);  
//Встановимо кінцеву точку  
line.setEndX(100.0f);  
line.setEndY(100.0f);
```

Таким чином буде створений прями відрізок від точки (0,0) до (100,100).

### **Прямокутник**

```
Rectangle r = new Rectangle(50,50,200,100);
```

або

```
Rectangle r = new Rectangle();  
//Встановимо верхній лівий кут  
r.setX(50);  
r.setY(50);  
//Встановимо розміри  
r.setWidth(200);  
r.setHeight(100);
```

Також можна заокруглити кути прямокутника.

```
r.setArcWidth(20);  
r.setArcHeight(20);
```

### Коло

```
Circle circle = new Circle(100,100,50);
```

або

```
Circle circle = new Circle();  
circle.setCenterX(100.0f);  
circle.setCenterY(100.0f);  
circle.setRadius(50.0f);
```

### Багатокутник (полігон)

```
Polygon polygon = new Polygon(0, 0, 20, 10, 10, 20);
```

або

```
Polygon polygon = new Polygon();  
polygon.getPoints().addAll(new Double[] {  
    0.0, 0.0,  
    20.0, 10.0,  
    10.0, 20.0 }));
```

або

```
polygon.getPoints().add(0.0);  
polygon.getPoints().add(0.0);  
polygon.getPoints().add(20.0);  
polygon.getPoints().add(10.0);  
polygon.getPoints().add(10.0);  
polygon.getPoints().add(20.0);
```

Зверніть увагу – в той час як конструктор приймає послідовність значень примітивного типу *double*, методи *add* та *addAll* працюють з об'єктами типу *Double*. Це означає, що до цих методів не можна передавати *int* та *float*, тобто виклики

```
polygon.getPoints().add(5);
```

та

```
polygon.getPoints().add(5.0f) ;
```

не спрацюють і компілятор повідомить про помилку.

Також важливо звернути увагу на те, що хоча кількість дійсних чисел, що задають полігон, може бути довільною, вона має бути **парною** (X та Y координати точки) та їх кількість має бути не менше 6. Майте на увазі, якщо кількість дійсних чисел була меншою ніж 6, полігон не відобразиться, але

виключних ситуацій (exseption) та повідомлень компілятора про помилку не буде. Якщо ж кількість була непарною, останнє значення просто ігнорується.

### **Ламана крива**

```
Polyline polyline = new Polyline(0.0, 0.0, 20.0, 10.0, 10.0, 20.0);
```

або

```
Polyline polyline = new Polyline();  
polyline.getPoints().addAll(new Double[] {  
    0.0, 0.0,  
    20.0, 10.0,  
    10.0, 20.0 }));
```

Загалом ті ж зауваження, що й для полігону, однак оскільки відображаються відрізки а не площі, мінімальна кількість дійсних чисел, що задають фігуру, може складати 4 (2 точки по 2 координати).

### **Еліпс**

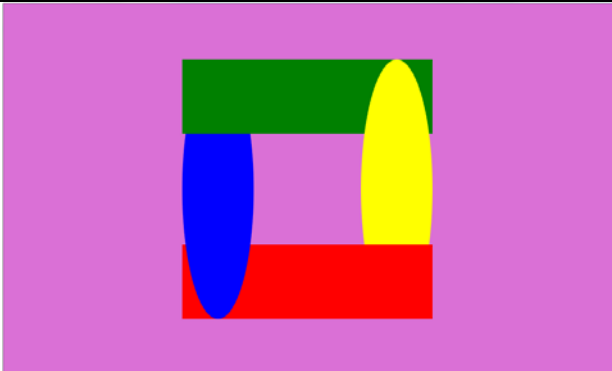
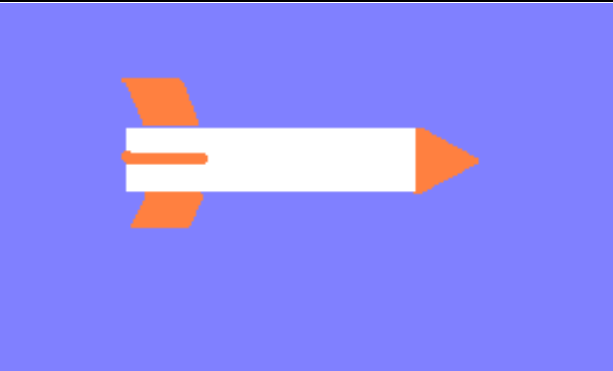
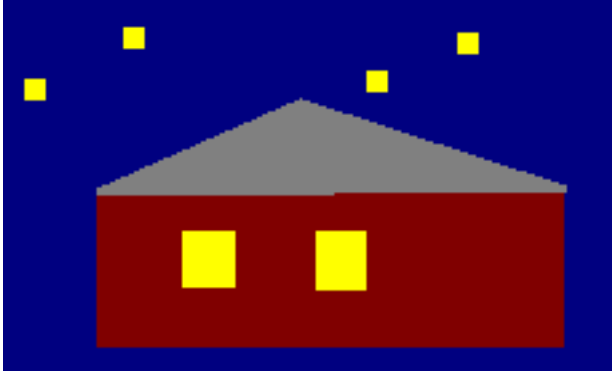
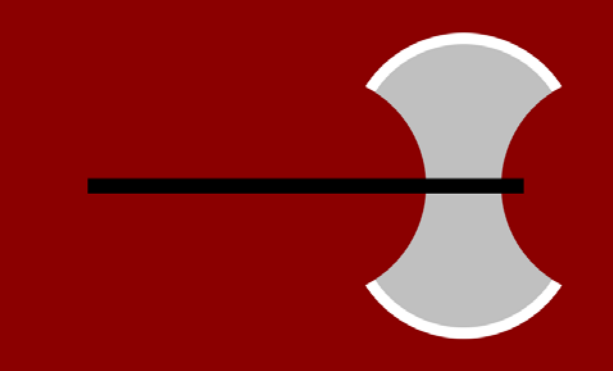

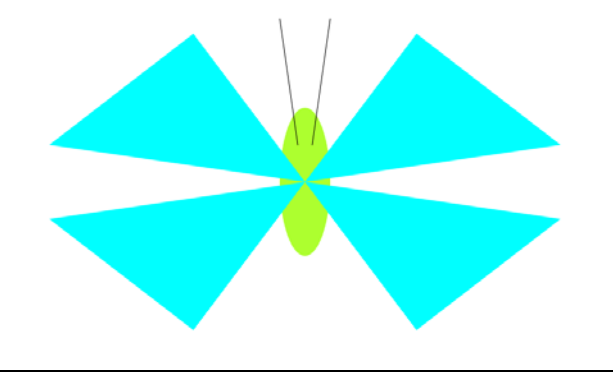
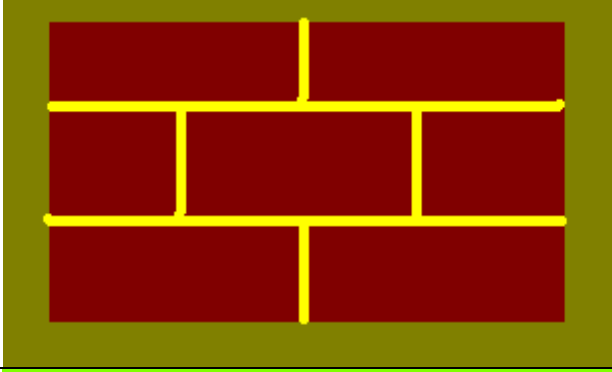
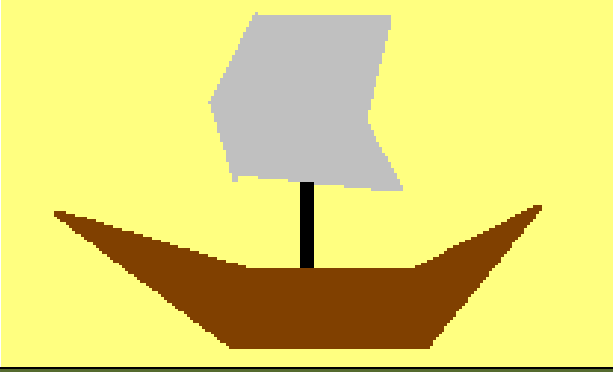
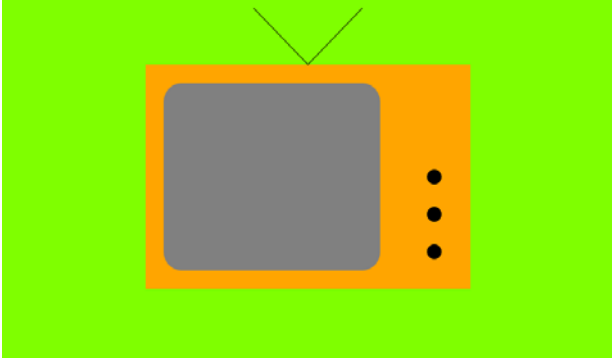

```
Ellipse ellipse = new Ellipse(50,50,50,25);
```

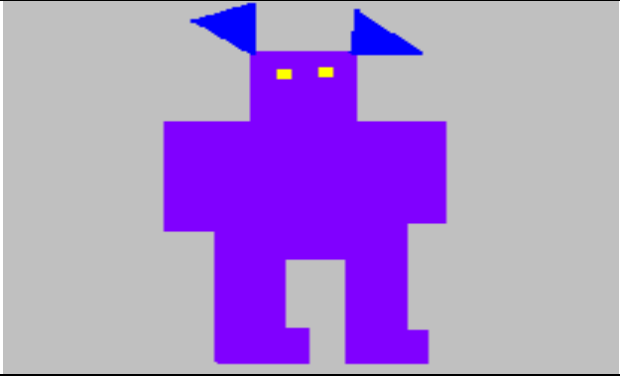
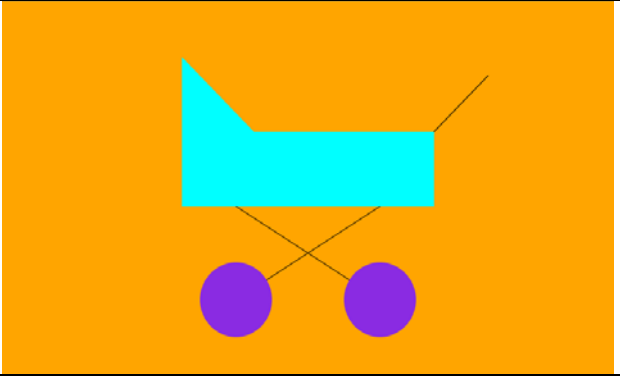
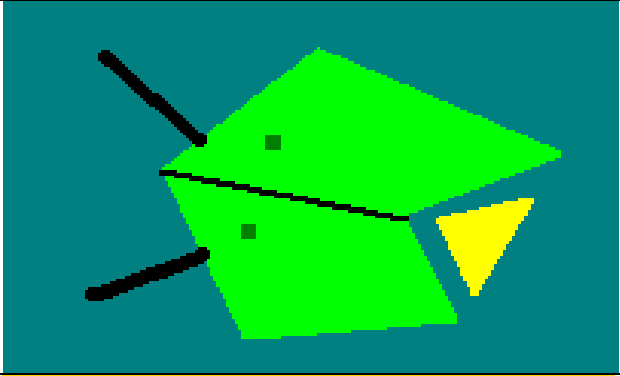

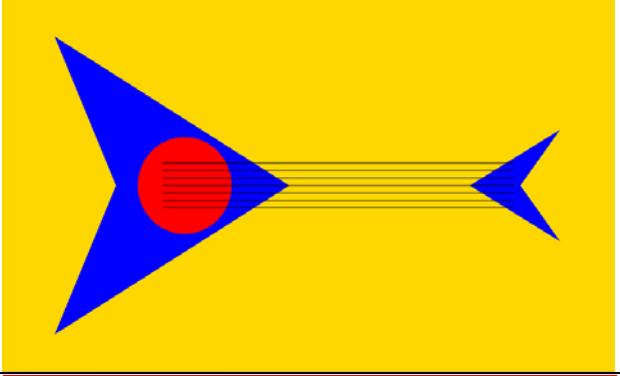
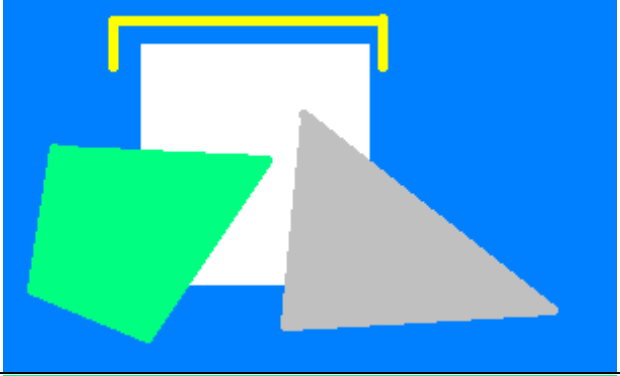

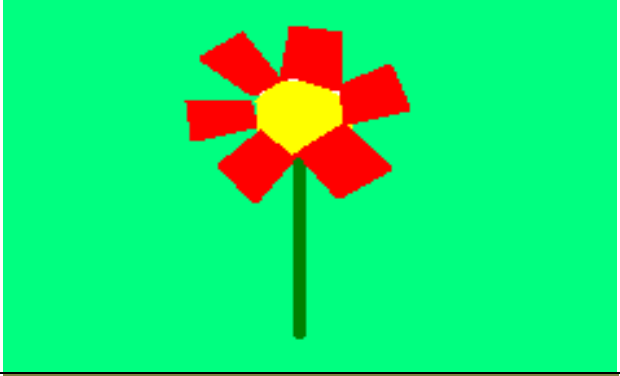
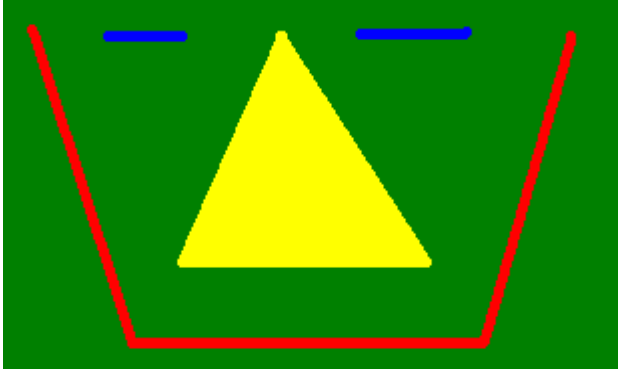
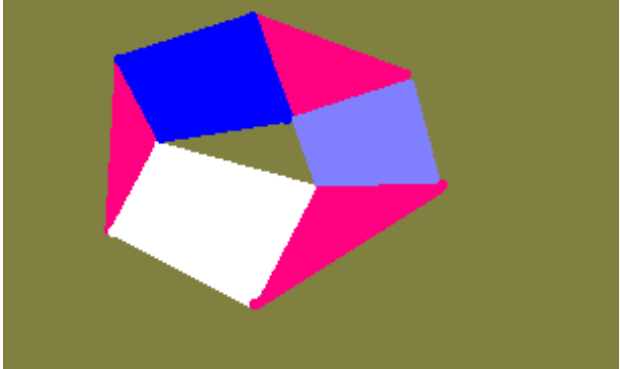
або

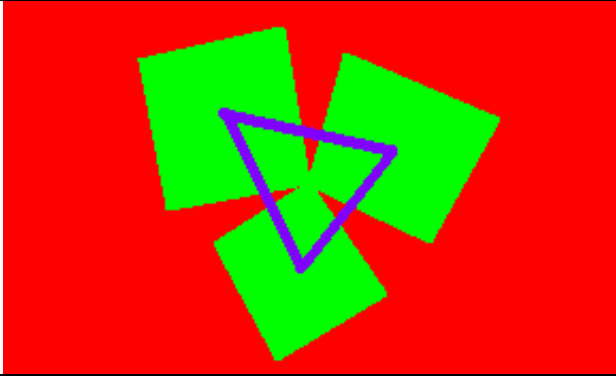
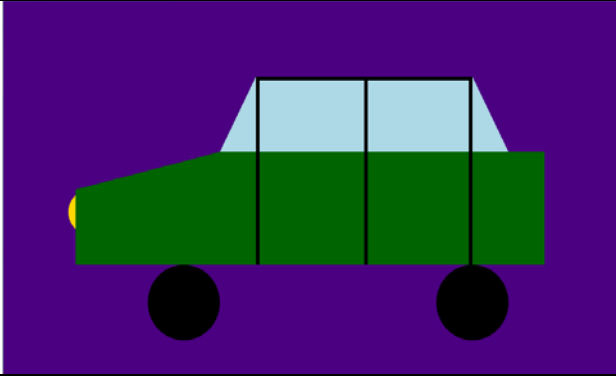

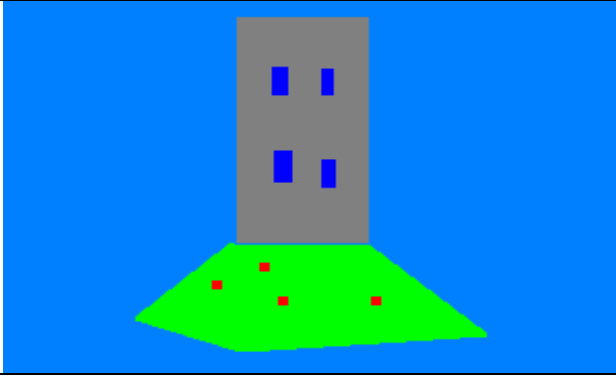
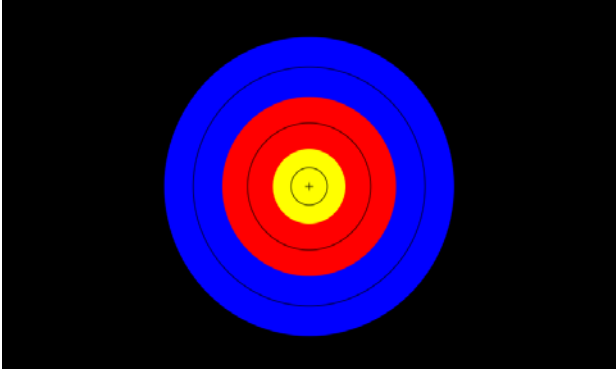
```
Ellipse ellipse = new Ellipse();  
ellipse.setCenterX(50.0f);  
ellipse.setCenterY(50.0f);  
ellipse.setRadiusX(50.0f);  
ellipse.setRadiusY(25.0f);
```



# Варианти завдань:

1.		14.	
2.		15.	
3.		16.	
4.		17.	
5.		18.	

6.		19.	
7.		20.	
8.		21.	
9.		22.	
10.		23.	

11.		24.	
12.		25.	
13.			

### Рекомендації до виконання

Для того, щоб точно підібрати колір, можна скористатися безкоштовною програмою ColorPic, яка показує колір пікселя під курсором миші та його представлення в кольоровій моделі RGB (для розповсюджених кольорів також відображається назва).

