# TASK DESCRIPTION

This program basically shows the traffic light system with GUI having three lights namely green, yellow, red which turns to their respective colour with user defined time. It uses multithreading process that handles changing colour feature of each light. Whole program is completed by following steps:

1. First, stage layout is created
2. Stack pane GUI component is added so that lights can be set into it.
3. Then rectangle is created that will work as a background for lights in black colour.
4. Green light is created with default grey colour property.
5. Yellow light is created with default grey colour property.
6. Red light is created with default grey colour property.
7. Then, flow pane is created and all the circle are stored in it with some default properties.
8. Both, flow pane and rectangle are set into stack pane.
9. Grid pane is created to store input objects.
10. Then, label and text field for green, yellow and red light are created with basic properties and set into grid pane.
11. Then, start and stop buttons are created and added to the grid pane.
12. Event handler for start button is used which takes the text field value input by use as time value for each light and valid number is checked.
13. Then, thread is created and runs for all the lights according to time value. Sets colour and prints the statement for each iteration.
14. Then, event handler for stop button is used which interrupts the thread and program without closing the window.
15. Then, Vertical box is created in which stack pane and grid pane is added.
16. V box is added to scene of specific size.
17. Scene is added to stage.

# TASK OUTPUT

|  |  |
| --- | --- |
| **Test Data** | **Screenshot** |
| Runs with default value  Start time – 3 for all lights |  |
| Runs with default value  Start time – 3 for all lights but interrupted in between with stop button |  |
| Runs with changed value  Start time:  Green: 5  Yellow: 3  Red: 2 |  |

# TASK CODE

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.geometry.Insets;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Alert;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.GridPane;

**import** javafx.scene.layout.StackPane;

**import** javafx.scene.layout.FlowPane;

**import** javafx.scene.layout.VBox;

**import** javafx.stage.Stage;

**import** javafx.scene.shape.Rectangle;

**import** javafx.scene.shape.Circle;

**import** javafx.scene.paint.Color;

**import** java.lang.Runnable;

**public** **class** TrafficLights **extends** Application {

// declaring thread variable

**private** Thread thread;

@Override

**public** **void** start(Stage stage) **throws** Exception

{

// Create stackpane to set traffic lights with default properties

StackPane stackPane = **new** StackPane();

stackPane.setAlignment(Pos.***TOP\_CENTER***);

// creating rectangle to set background for traffic lights with default properties

Rectangle rectangle = **new** Rectangle(10, 10, 450, 200);

rectangle.setFill(Color.***BLACK***);

rectangle.setArcHeight(50);

rectangle.setArcWidth(50);

// creating green circle

Circle green = **new** Circle(10,10,50);

green.setFill(Color.***DARKGRAY***);

//creating yellow circle

Circle yellow = **new** Circle(10,10,50);

yellow.setFill(Color.***DARKGRAY***);

// creating red circle

Circle red = **new** Circle(10,10,50);

red.setFill(Color.***DARKGRAY***);

// create flowpane and store all the circles in it

FlowPane flowPane = **new** FlowPane();

flowPane.setPadding(**new** Insets(50, 20, 50, 20));

flowPane.setHgap(50);

flowPane.getChildren().addAll(green, yellow, red);

flowPane.setAlignment(Pos.***TOP\_CENTER***);

// set rectangle and flowpane in stackpane with default properties

stackPane.getChildren().addAll(rectangle,flowPane);

stackPane.setPadding(**new** Insets(15,0,0,0));

// Create gridpane to store textfields and labels

GridPane gridPane = **new** GridPane();

gridPane.setHgap(20);

gridPane.setVgap(20);

gridPane.setPadding(**new** Insets(20,20,20,20));

gridPane.setAlignment(Pos.***TOP\_CENTER***);

// creating label and textfield to control green light with default properties

Label labelGreen = **new** Label("Green");

TextField textFieldGreen = **new** TextField();

gridPane.add(labelGreen, 0, 0);

gridPane.add(textFieldGreen, 1, 0);

textFieldGreen.setText("3");

textFieldGreen.setAlignment(Pos.***BOTTOM\_LEFT***);

// creating label and textfield to control yellow light with default properties

Label labelYellow = **new** Label("Yellow");

TextField textFieldYellow = **new** TextField();

gridPane.add(labelYellow, 0, 1);

gridPane.add(textFieldYellow, 1, 1);

textFieldYellow.setText("3");

textFieldYellow.setAlignment(Pos.***BOTTOM\_LEFT***);

// creating label and textfield to control red light with default properties

Label labelRed = **new** Label("Red");

TextField textFieldRed = **new** TextField();

gridPane.add(labelRed, 0, 2);

gridPane.add(textFieldRed, 1, 2);

textFieldRed.setText("3");

textFieldRed.setAlignment(Pos.***BOTTOM\_LEFT***);

// creating start and stop button and setting its position in gridpane

Button buttonStart = **new** Button("Start");

gridPane.add(buttonStart, 2, 2);

Button buttonStop = **new** Button("Stop");

gridPane.add(buttonStop, 3, 2);

// setting event handler to start button

buttonStart.setOnAction(**new** EventHandler<ActionEvent>()

{

@Override

**public** **void** handle(ActionEvent e)

{

// getting the values of all light textfields and parsing into integer

String greenLightTime = textFieldGreen.getText();

String yellowLightTime = textFieldYellow.getText();

String redLightTime = textFieldRed.getText();

// checks and validates light time input by user and displays message accordingly

**if** (!*isPositiveInteger*(greenLightTime) || !*isPositiveInteger*(yellowLightTime) || !*isPositiveInteger*(redLightTime))

{

System.***out***.println("Problem occured.!");

// creates alert if any error in input

Alert alert = **new** Alert(Alert.AlertType.***ERROR***);

alert.setTitle("INVALID TIME INPUT");

alert.setHeaderText(**null**);

alert.setContentText("Only positive integers are allowed");

alert.showAndWait();

}

**else**

{

buttonStart.setDisable(**true**); // start button is disabled

// creates a new thread

thread = **new** Thread(**new** Runnable()

{

@Override

**public** **void** run()

{

**try**

{

**int** timeCount = 0;

**for** (**int** i = 0; i < 3; i++)

{

System.***out***.println("change light color");

// using switch case to switch between lights

**switch** (i)

{

// case for green light

**case** 0:

// sets green color to first light

green.setFill(Color.***GREEN***);

timeCount = Integer.*parseInt*(greenLightTime);

**break**;

// case for yellow light

**case** 1:

// sets yellow color to second light

yellow.setFill(Color.***YELLOW***);

timeCount = Integer.*parseInt*(yellowLightTime);

**break**;

// case for red light

**case** 2:

// sets red color to third light

red.setFill(Color.***RED***);

timeCount = Integer.*parseInt*(redLightTime);

**break**;

}

// using for loop to continue light for given input

**for** (**int** j = 0; j < timeCount; j++)

{

System.***out***.println("now light: " + String.*valueOf*(i+1) + " after "+ String.*valueOf*(timeCount-j-1) + " seconds will change color");

Thread.*sleep*(1000);

}

// sets default color

green.setFill(Color.***GREY***);

yellow.setFill(Color.***GREY***);

red.setFill(Color.***GREY***);

}

}

// catches execption if there is problem during current thread

**catch** (InterruptedException ex)

{

Thread.*currentThread*().interrupt();

}

buttonStart.setDisable(**false**); // start button is enabled

}

});

thread.setDaemon(**true**);

thread.start();

}

}

});

// setting event handler to stop button

buttonStop.setOnAction(**new** EventHandler<ActionEvent>()

{

@Override

**public** **void** handle(ActionEvent e)

{

buttonStart.setDisable(**false**); // start button is enabled

// checks if thread is not null and is alive

**if** (thread != **null** && thread.isAlive())

{

thread.interrupt(); // stops thread

System.***out***.println("TRAFFIC LIGHTS HAS BEEN STOPPED.!");

}

// setting default colors to lights

green.setFill(Color.***GREY***);

yellow.setFill(Color.***GREY***);

red.setFill(Color.***GREY***);

}

});

// creating vertical box and setting stackpane and gridpane in it with some default properties

VBox vbox = **new** VBox();

vbox.setPadding(**new** Insets(10,10,10,10));

vbox.getChildren().addAll(stackPane,gridPane);

// creating scene and setting vbox in it with some default properties

Scene scene = **new** Scene(vbox, 500, 380);

stage.setTitle("Traffic light system");

stage.setResizable(**false**); // fix the size of stage

stage.setScene(scene); // setting scene in the primary stage

stage.show(); // show the stage

}

// method to check and validate correct input with only positive integers allowed

**public** **static** **boolean** isPositiveInteger(String input)

{

// checks if input in numeric

**try**

{

Integer.*parseInt*(input);

}

**catch** (NumberFormatException e)

{

**return** **false**;

}

// checks if input is positive integer

**if** (Integer.*parseInt*(input) >= 0)

{

**return** **true**;

}

**else**

{

**return** **false**;

}

}

**public** **static** **void** main(String[] args)

{

*launch*();

}

}