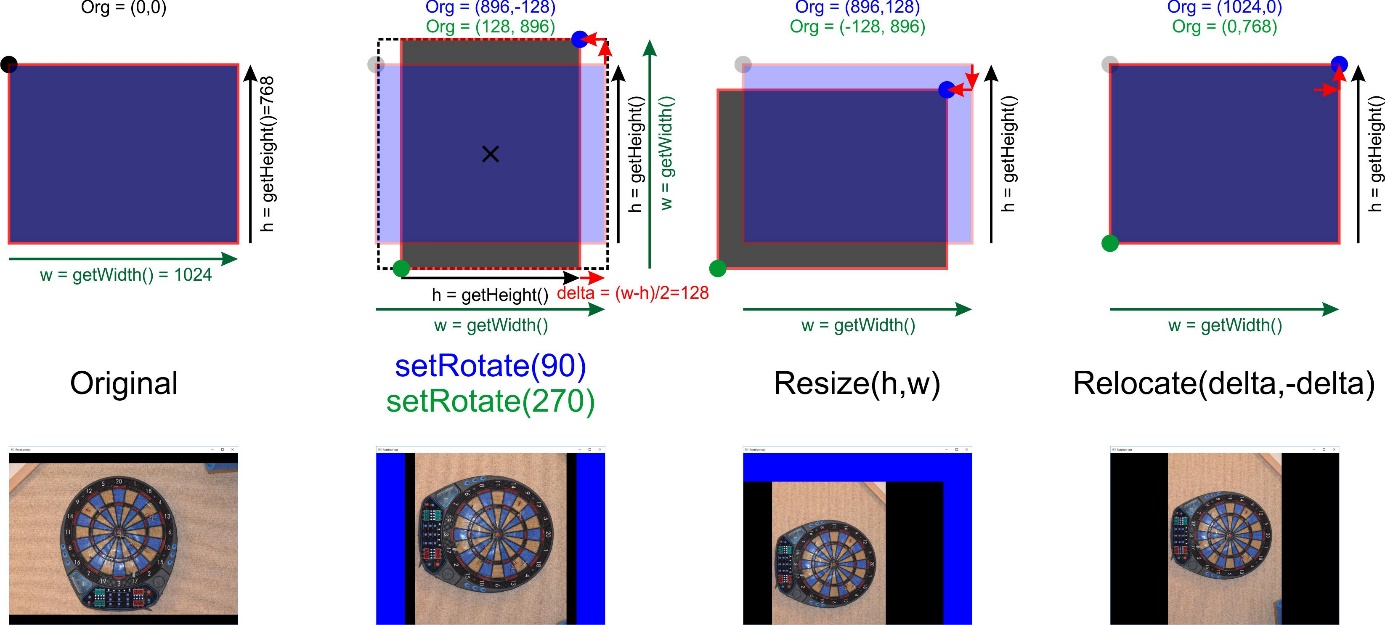
[https://stackoverflow.com/questions/53109791/fitting-rotated-imageview-into-application-window-scene/53308535#53308535](https://stackoverflow.com/questions/53109791/fitting-rotated-imageview-into-application-window-scene/53308535" \l "53308535)

I found a solution :-) Fabian's approach inspired me (thank you!!) And my old friend Pit helped me with debugging (also thank you!!)

It seems that the layout location algorithm of JavaFX has a problem when resize() is applied to rotated Panes (or even Nodes - I have not tried):

[](https://i.stack.imgur.com/fjyB8.jpg)

Following Fabian's idea I debugged into the layoutChildren() method of class Pane. I found that the relocation after setRotate() is correct and keeps the center of the child pane as expected. But as soon as resize() is called (which is done because of fitting the rotated child pane again into its father and additionally always when the window is resized by the user) the origin calculation goes wrong:

The picture above depicts a sequence of setRotate(90), resize() and relocate() in green and the same for setRotate(270) in blue. A little blue/green circle depicts the corresponding origin together with its coordinates in the 1024x786 example.

**Analysis**

It seems that for calculation the position of the Pane resize() does not use the height and width from BoundsInParent-Property (see JavaFX-Docu of Node) but from getWidth() and getHeight() which seem to reflect BoundsInLocal. As a consequence, for rotations of 90° or 270° height and width seem to be interchanged. Therefore the error in the calculation for the new origin is just the half of the difference between width and height (delta=(width-height)/2) when resize() tries to center the child pane again after the resizing.

**Solution**

A relocation(delta,-delta) needs to be applied after resizing for Panes with rotation=90 or 270 degrees.

The structure of my implementation follows Fabian's basic idea: I have build a layouter RotatablePaneLayouter:Region that just overwrites the layoutChildren() method. In its constructor it gets a Pane (in my example a StackPane) which can contain any number of children (in my example an ImageView) and that can be rotated.

LayoutChildren() then just executes resize() and relocate() for the child pane to fit it completely into the RotateablePaneLayouter respecting the orientation of the child pane.

**The Layouter Helper (RotateablePaneLayouter:Region)**

public class RotatablePaneLayouter extends Region {

private Pane child;

public RotatablePaneLayouter(Pane child) {

getChildren().add(child);

this.child = child;

// make sure layout gets invalidated when the child orientation changes

child.rotateProperty().addListener(new ChangeListener<Number>() {

@Override

public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number newValue) {

requestLayout();

}

});

}

@Override

protected void layoutChildren() {

// set fit sizes:

//resize child to fit into RotatablePane and correct movement caused by resizing if necessary

if ((child.getRotate() == 90)||(child.getRotate() == 270)) {

//vertical

child.resize( getHeight(), getWidth() ); //exchange width and height

// and relocate to correct movement caused by resizing

double delta = (getWidth() - getHeight()) / 2;

child.relocate(delta,-delta);

} else {

//horizontal

child.resize( getWidth(), getHeight() ); //keep width and height

//with 0° or 180° resize does no movement to be corrected

child.relocate(0,0);

}

}

}

To use it: Place the Pane to be rotated into the Layouter first instead of placing the Pane directly.

Here the code for the example's main program. You can use the space bar to rotate the child pane by 90, 180, 270 and again 0 degrees. You can also resize the window with the mouse. The layouter always manages to place the rotated pane correctly.

**Expample for using the Layouter**

public class RotationTest extends Application {

public static void main(String[] args) {

Application.launch(args);

}

@Override

public void start(Stage primaryStage) {

//image in a StackPane to be rotated

final ImageView imageView = new ImageView("file:D:/Test\_org.jpg");

imageView.setPreserveRatio(true);

StackPane stackPane = new StackPane(imageView); //a stackPane is used to center the image

stackPane.setStyle("-fx-background-color: black;");

imageView.fitWidthProperty().bind(stackPane.widthProperty());

imageView.fitHeightProperty().bind(stackPane.heightProperty());

//container for layouting rotated Panes

RotatablePaneLayouter root = new RotatablePaneLayouter(stackPane);

root.setStyle("-fx-background-color: blue;");

Scene scene = new Scene(root, 1024,768);

scene.setOnKeyPressed(new EventHandler<KeyEvent>() {

@Override

public void handle(KeyEvent event) {

if (event.getCode() == KeyCode.SPACE) {

//rotate additionally 90°

stackPane.setRotate((stackPane.getRotate() + 90) % 360);

}

}

});

primaryStage.setTitle("Rotation test");

primaryStage.setScene(scene);

primaryStage.show();

}

}

For me this seems like a workaround of a javaFX bug in resize().

Hier der Rohtext, wie er hochgeladen wurde (und dann gelayoutet durch Stackoverflow)

Hello Everyone

I found a solution :-) Fabian's approach inspired me (thank you!!) And my old friend Pit helped me with debugging (also thank you!!)

It seems that the layout location algorithm of JavaFX has a problem when resize() is applied to rotated Panes (or even Nodes - I have not tried):

[![Origin movement following setRotate and resize][1]][1]

[1]: https://i.stack.imgur.com/fjyB8.jpg

Following Fabian's idea I debugged into the layoutChildren() method of class Pane. I found that the relocation after setRotate() is correct and keeps the center of the child pane as expected. But as soon as resize() is called (which is done because of fitting the rotated child pane again into its father and additionally always when the window is resized by the user) the origin calculation goes wrong:

The picture above depicts a sequence of setRotate(90), resize() and relocate() in green and the same for setRotate(270) in blue. A little blue/green circle depicts the corresponding origin together with its coordinates in the 1024x786 example.

## Analysis ##

It seems that for calculation the position of the Pane resize() does not use the height and width from BoundsInParent-Property (see JavaFX-Docu of Node) but from getWidth() and getHeight() which seem to reflect BoundsInLocal. As a consequence, for rotations of 90° or 270° height and width seem to be interchanged. Therefore the error in the calculation for the new origin is just the half of the difference between width and height (delta=(width-height)/2) when resize() tries to center the child pane again after the resizing.

## Solution ##

A relocation(delta,-delta) needs to be applied after resizing for Panes with rotation=90 or 270 degrees.

The structure of my implementation follows Fabian's basic idea: I have build a layouter RotatablePaneLayouter:Region that just overwrites the layoutChildren() method. In its constructor it gets a Pane (in my case StackPane) that can be rotated. LayoutChildren() then just executes resize() and relocate() for the child pane to fit it completely into the RotateablePaneLayouter respecting the orientation of the child pane.

## The Layouter Helper (RotateablePaneLayouter:Region) ##

public class RotatablePaneLayouter extends Region {

private Pane child;

public RotatablePaneLayouter(Pane child) {

getChildren().add(child);

this.child = child;

// make sure layout gets invalidated when the child orientation changes

child.rotateProperty().addListener(new ChangeListener<Number>() {

@Override

public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number newValue) {

requestLayout();

}

});

}

@Override

protected void layoutChildren() {

// set fit sizes:

//resize child to fit into RotatablePane and correct movement caused by resizing if necessary

if ((child.getRotate() == 90)||(child.getRotate() == 270)) {

//vertical

child.resize( getHeight(), getWidth() ); //exchange width and height

// and relocate to correct movement caused by resizing

double delta = (getWidth() - getHeight()) / 2;

child.relocate(delta,-delta);

} else {

//horizontal

child.resize( getWidth(), getHeight() ); //keep width and height

//with 0° or 180° resize does no movement to be corrected

child.relocate(0,0);

}

}

}

To use it: Place the Pane to be rotated into the Layouter first instead of placing the Pane directly.

Here the code for the main progam. You can use the space bar to rotate the child pane by 90, 180, 270 and again 0 degrees. You can also resize the window with the mouse. The layouter manages to place the rotated pane correctly.

## Expample for using the Layouter ##

public class RotationTest extends Application {

public static void main(String[] args) {

Application.launch(args);

}

@Override

public void start(Stage primaryStage) {

//image in a StackPane to be rotated

final ImageView imageView = new ImageView("file:D:/Test\_org.jpg");

imageView.setPreserveRatio(true);

StackPane stackPane = new StackPane(imageView); //a stackPane is used to center the image

stackPane.setStyle("-fx-background-color: black;");

imageView.fitWidthProperty().bind(stackPane.widthProperty());

imageView.fitHeightProperty().bind(stackPane.heightProperty());

//container for layouting rotated Panes

RotatablePaneLayouter root = new RotatablePaneLayouter(stackPane);

root.setStyle("-fx-background-color: blue;");

Scene scene = new Scene(root, 1024,768);

scene.setOnKeyPressed(new EventHandler<KeyEvent>() {

@Override

public void handle(KeyEvent event) {

if (event.getCode() == KeyCode.SPACE) {

//rotate additionally 90°

stackPane.setRotate((stackPane.getRotate() + 90) % 360);

}

}

});

primaryStage.setTitle("Rotation test");

primaryStage.setScene(scene);

primaryStage.show();

}

}

For me this seems like a workaround of a javaFX bug in resize().