JavaFX

Oracle JavaFX Applying Transformations in JavaFX

Release 2.1

E20471-03

October 2012

ORACLE®

JavaFX Applying Transformations in JavaFX, Release 2.1

E20471-03

Copyright © 2011, 2012 Oracle and/or its affiliates. All rights reserved.

Primary Author: Dmitry Kostovaorv

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

1 Translation

2 Rotation

3 Scaling

4 Shearing

5 Multiple Transformations

Part I

About This Document

A transformation changes the place of a graphical object in a coordinate system according to certain parameters. The following types of transformations are supported in JavaFX:

- Translation
- Rotation
- Scaling
- Shearing

These transformations can be applied to either a standalone node or to groups of nodes. You can apply one transformation at a time or you can combine transformations and apply several transformations to one node.

All transformations are located in the `javafx.scene.transform` package and are subclasses of the `Transform` class.

The `Transform` class implements the concepts of affine transformations. The `Affine` class extends the `Transform` class and acts as a superclass to all transformations. Affine transformations are based on euclidean algebra, and perform a linear mapping (through the use of matrixes) from initial coordinates to other coordinates while preserving the straightness and parallelism of lines. Affine transformations can be constructed using observableArrayLists rotations, translations, scales, and shears.

Note: Usually, do not use the `Affine` class directly, but instead, use the specific Translate, Scale, Rotate, or Shear transformations.

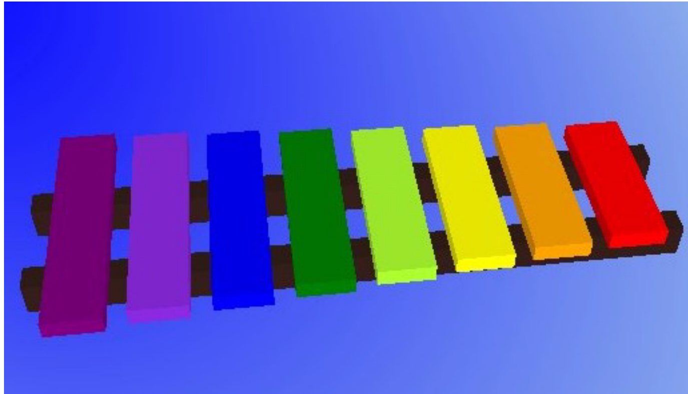
Transformations in JavaFX can be performed along three coordinates, thus enabling users to create three-dimensional (3-D) objects and effects. To manage the display of objects with depth in 3-D graphics, JavaFX implements z-buffering. Z-buffering ensures that the perspective is the same in the virtual world as it is in the real one: a solid object in the foreground blocks the view of one behind it. Z-buffering can be enabled by using the `setDepthTest` class. You can try to disable z-buffering (`setDepthTest(DepthTest.DISABLE)`) in the sample application to see the effect of the z-buffer.

To simplify transformation usage, JavaFX implements transformation constructors with the x-axis and y-axis along with the x, y, and z axes. If you want to create a two-dimensional (2-D) effect, you can specify only the x and y coordinates. If you want to create a 3-D effect, specify all three coordinates.

To be able to see 3-D objects and transformation effects in JavaFX, users must enable the perspective camera.

Though knowing the underlying concepts can help you use JavaFX more effectively, you can start using transformations by studying the example provided with this document and trying different transformation parameters. For more information about particular classes, methods, or additional features, see the API documentation.

In this document, a Xylophone application is used as a sample to illustrate all the available transformations.



1 Translation

The translation transformation shifts a node from one place to another along one of the axes relative to its initial position. The initial position of the xylophone bar is defined by x, y, and z coordinates. In Example 1–1, the initial position values are specified by the `xStart`, `yPos`, and `zPos` variables. Some other variables are added to simplify the calculations when applying different transformations. Each bar of the xylophone is based on one of the base bars. The example then translates the base bars with different shifts along the three axes to correctly locate them in space.

Example 1–1 shows a code snippet from the sample application with the translation transformation.

Example 1–1 Translation

```
Group rectangleGroup = new Group();
rectangleGroup.setDepthTest(DepthTest.ENABLE);

double xStart = 260.0;
double xOffset = 30.0;
double yPos = 300.0;
double zPos = 0.0;
double barWidth = 22.0;
double barDepth = 7.0;

// Base1
Cube base1Cube = new Cube(1.0, new Color(0.2, 0.12, 0.1, 1.0), 1.0);
base1Cube.setTranslateX(xStart + 135);
base1Cube.setTranslateZ(yPos+20.0);
base1Cube.setTranslateY(11.0);
```

2

Rotation

The rotation transformation moves the node around a specified pivot point of the scene. You can use the `rotate` function of the `Transform` class to perform the rotation.

To rotate the camera around the xylophone in the sample application, the rotation transformation is used, although technically, it is the xylophone itself that is moving when the mouse rotates the camera.

Example 2–1 shows the code for the rotation transformation.

Example 2–1 Rotation

```
    class Cam extends Group {
        Translate t  = new Translate();
        Translate p  = new Translate();
        Translate ip = new Translate();
        Rotate rx = new Rotate();
        { rx.setAxis(Rotate.X_AXIS); }
        Rotate ry = new Rotate();
        { ry.setAxis(Rotate.Y_AXIS); }
        Rotate rz = new Rotate();
        { rz.setAxis(Rotate.Z_AXIS); }
        Scale s = new Scale();
        public Cam() { super(); getTransforms().addAll(t, p, rx, rz, ry, s, ip); }
    }
...
        scene.setOnMouseDragged(new EventHandler<MouseEvent>() {
            public void handle(MouseEvent me) {
                mouseOldX = mousePosX;
                mouseOldY = mousePosY;
                mousePosX = me.getX();
                mousePosY = me.getY();
                mouseDeltaX = mousePosX - mouseOldX;
                mouseDeltaY = mousePosY - mouseOldY;
                if (me.isAltDown() && me.isShiftDown() &&
me.isPrimaryButtonDown()) {
                    cam.rz.setAngle(cam.rz.getAngle() - mouseDeltaX);
                }
                else if (me.isAltDown() && me.isPrimaryButtonDown()) {
                    cam.ry.setAngle(cam.ry.getAngle() - mouseDeltaX);
                    cam.rx.setAngle(cam.rx.getAngle() + mouseDeltaY);
                }
                else if (me.isAltDown() && me.isSecondaryButtonDown()) {
                    double scale = cam.s.getX();
                    double newScale = scale + mouseDeltaX*0.01;
                    cam.s.setX(newScale); cam.s.setY(newScale);
cam.s.setZ(newScale);
```

```
            }
            else if (me.isAltDown() && me.isMiddleButtonDown()) {
                cam.t.setX(cam.t.getX() + mouseDeltaX);
                cam.t.setY(cam.t.getY() + mouseDeltaY);
            }
        }
    });
```

Note that the pivot point and the angle define the destination point the image is moved to. Carefully calculate values when specifying the pivot point. Otherwise, the image might appear where it is not intended to be. For more information, see the API documentation.

3

Scaling

The scaling transformation causes a node to either appear larger or smaller, depending on the scaling factor. Scaling changes the node so that the dimensions along its axes are multiplied by the scale factor. Similar to the rotation transformations, scaling transformations are applied at a pivot point. This pivot point is considered the point around which scaling occurs.

To scale, use the `Scale` class and the `scale` function of the `Transform` class.

In the Xylophone application, you can scale the xylophone using the mouse while pressing Alt and the right mouse button. The scale transformation is used to see the scaling.

Example 3–1 shows the code for the scale transformation.

Example 3–1 Scaling

```
      else if (me.isAltDown() && me.isSecondaryButtonDown()) {
          double scale = cam.s.getX();
          double newScale = scale + mouseDeltaX*0.01;
          cam.s.setX(newScale); cam.s.setY(newScale); cam.s.setZ(newScale);
                }
...
```

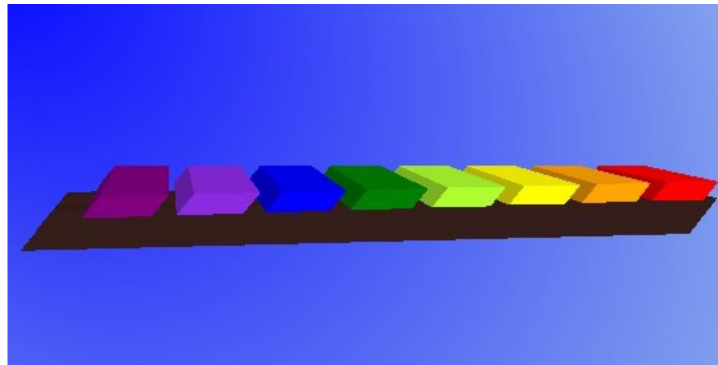
4 Shearing

A shearing transformation rotates one axis so that the x-axis and y-axis are no longer perpendicular. The coordinates of the node are shifted by the specified multipliers.

To shear, use the `Shear` class or the `shear` function of the `Transform` class.

In the Xylophone application, you can shear the xylophone by dragging the mouse while holding Shift and pressing the left mouse button.

Figure 4–1 Shearing Transformation



Example 4–1 shows the code snippet for the shear transformation.

Example 4–1 Shearing

```
else if (me.isShiftDown() && me.isPrimaryButtonDown()) {
    double yShear = shear.getY();
    shear.setY(yShear + mouseDeltaY/1000.0);
    double xShear = shear.getX();
    shear.setX(xShear + mouseDeltaX/1000.0);
}
```

5

Multiple Transformations

You can construct multiple transformations by specifying an ordered chain of transformations. For example, you can scale an object and then apply a shearing transformation to it, or you can translate an object and then scale it.

Example 5–1 shows multiple transformations applied to an object to create a xylophone bar.

Example 5–1 Multiple Transformation

```
Cube base1Cube = new Cube(1.0, new Color(0.2, 0.12, 0.1, 1.0), 1.0);
base1Cube.setTranslateX(xStart + 135);
base1Cube.setTranslateZ(yPos+20.0);
base1Cube.setTranslateY(11.0);
base1Cube.setScaleX(barWidth*11.5);
base1Cube.setScaleZ(10.0);
base1Cube.setScaleY(barDepth*2.0);
```