```
/*
 * Copyright (c) 2010, 2014, Oracle and/or its affiliates. All rights reserved.
 * ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */

package javafx.scene.chart;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import javafx.animation.Animation;
import javafx.animation.FadeTransition;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.DoublePropertyBase;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.ObjectPropertyBase;
import javafx.beans.property.ReadOnlyObjectProperty;
import javafx.beans.property.ReadOnlyObjectWrapper;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.property.StringPropertyBase;
import javafx.beans.value.WritableValue;
import javafx.collections.FXCollections;
import javafx.collections.ListChangeListener;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.NodeOrientation;
import javafx.geometry.Side;
import javafx.scene.AccessibleRole;
import javafx.scene.Node;
import javafx.scene.layout.Region;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcTo;
import javafx.scene.shape.ArcType;
import javafx.scene.shape.ClosePath;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.text.Text;
import javafx.scene.transform.Scale;
import javafx.util.Duration;

```

```
72  import com.sun.javafx.charts.Legend;
73  import com.sun.javafx.charts.Legend.LegendItem;
74  import com.sun.javafx.collections.NonIterableChange;
75
76  import javafx.css.StyleableBooleanProperty;
77  import javafx.css.StyleableDoubleProperty;
78  import javafx.css.CssMetaData;
79
80  import com.sun.javafx.css.converters.BooleanConverter;
81  import com.sun.javafx.css.converters.SizeConverter;
82  import java.util.BitSet;
83
84  import javafx.css.Styleable;
85  import javafx.css.StyleableProperty;
86
87  /**
88   * Displays a PieChart. The chart content is populated by pie slices based on
89   * data set on the PieChart.
90   * <p> The clockwise property is set to true by default, which means slices are
91   * placed in the clockwise order. The labelsVisible property is used to either display
92   * pie slice labels or not.
93   *
94   * @since JavaFX 2.0
95   */
96  public class PieChart extends Chart {
97
98      // -------------- PRIVATE FIELDS
        ----------------------------------------------------------------------------
        -
99      private static final int MIN_PIE_RADIUS = 25;
100     private static final double LABEL_TICK_GAP = 6;
101     private static final double LABEL_BALL_RADIUS = 2;
102     private BitSet colorBits = new BitSet(8);
103     private double centerX;
104     private double centerY;
105     private double pieRadius;
106     private Data begin = null;
107     private final Path labelLinePath = new Path() {
108         @Override public boolean usesMirroring() {
109             return false;
110         }
111     };
112     private Legend legend = new Legend();
113     private Data dataItemBeingRemoved = null;
114     private Timeline dataRemoveTimeline = null;
115     private final ListChangeListener<Data> dataChangeListener = c -> {
116         while (c.next()) {
117             // RT-28090 Probably a sort happened, just reorder the pointers.
118             if (c.wasPermutated()) {
119                 Data ptr = begin;
120                 for (int i = 0; i < getData().size(); i++) {
121                     Data item = getData().get(i);
122                     updateDataItemStyleClass(item, i);
123                     if (i == 0) {
124                         begin = item;
125                         ptr = begin;
126                         begin.next = null;
127                     } else {
128                         ptr.next = item;
129                         item.next = null;
130                         ptr = item;
131                     }
132                 }
133                 // update legend style classes
134                 if (isLegendVisible()) {
135                     updateLegend();
136                 }
137                 requestChartLayout();
138                 return;
139             }
140             // recreate linked list & set chart on new data
```

```
            for (int i = c.getFrom(); i < c.getTo(); i++) {
                Data item = getData().get(i);
                item.setChart(PieChart.this);
                if (begin == null) {
                    begin = item;
                    begin.next = null;
                } else {
                    if (i == 0) {
                        item.next = begin;
                        begin = item;
                    } else {
                        Data ptr = begin;
                        for (int j = 0; j < i -1 ; j++) {
                            ptr = ptr.next;
                        }
                        item.next = ptr.next;
                        ptr.next = item;
                    }
                }
            }
            // call data added/removed methods
            for (Data item : c.getRemoved()) {
                dataItemRemoved(item);
            }
            for (int i = c.getFrom(); i < c.getTo(); i++) {
                Data item = getData().get(i);
                // assign default color to the added slice
                // TODO: check nearby colors
                item.defaultColorIndex = colorBits.nextClearBit(0);
                colorBits.set(item.defaultColorIndex);
                dataItemAdded(item, i);
            }
            if (c.wasRemoved() || c.wasAdded()) {
                for (int i = 0; i < getData().size(); i++) {
                    Data item = getData().get(i);
                    updateDataItemStyleClass(item, i);
                }
                // update legend if any data has changed
                if (isLegendVisible()) {
                    updateLegend();
                }
            }
        }
        // re-layout everything
        requestChartLayout();
    };

    // -------------- PUBLIC PROPERTIES ----------------------------------------

    /** PieCharts data */
    private ObjectProperty<ObservableList<Data>> data = new ObjectPropertyBase<
    ObservableList<Data>>() {
        private ObservableList<Data> old;
        @Override protected void invalidated() {
            final ObservableList<Data> current = getValue();
            // add remove listeners
            if(old != null) old.removeListener(dataChangeListener);
            if(current != null) current.addListener(dataChangeListener);
            // fire data change event if series are added or removed
            if(old != null || current != null) {
                final List<Data> removed = (old != null) ? old : Collections.<Data>
                emptyList();
                final int toIndex = (current != null) ? current.size() : 0;
                // let data listener know all old data have been removed and new
                data that has been added
                if (toIndex > 0 || !removed.isEmpty()) {
                    dataChangeListener.onChanged(new NonIterableChange<Data>(0,
                    toIndex, current){
                        @Override public List<Data> getRemoved() { return removed; }
                        @Override public boolean wasPermutated() { return false; }
                        @Override protected int[] getPermutation() {
```

```
                            return new int[0];
                        }
                    });
                }
            } else if (old != null && old.size() > 0) {
                // let series listener know all old series have been removed
                dataChangeListener.onChanged(new NonIterableChange<Data>(0, 0,
                current){
                    @Override public List<Data> getRemoved() { return old; }
                    @Override public boolean wasPermutated() { return false; }
                    @Override protected int[] getPermutation() {
                        return new int[0];
                    }
                });
            }
            old = current;
        }

        public Object getBean() {
            return PieChart.this;
        }

        public String getName() {
            return "data";
        }
    };
    public final ObservableList<Data> getData() { return data.getValue(); }
    public final void setData(ObservableList<Data> value) { data.setValue(value); }
    public final ObjectProperty<ObservableList<Data>> dataProperty() { return data; }

    /** The angle to start the first pie slice at */
    private DoubleProperty startAngle = new StyleableDoubleProperty(0) {
        @Override public void invalidated() {
            get();
            requestChartLayout();
        }

        @Override
        public Object getBean() {
            return PieChart.this;
        }

        @Override
        public String getName() {
            return "startAngle";
        }

        public CssMetaData<PieChart,Number> getCssMetaData() {
            return StyleableProperties.START_ANGLE;
        }
    };
    public final double getStartAngle() { return startAngle.getValue(); }
    public final void setStartAngle(double value) { startAngle.setValue(value); }
    public final DoubleProperty startAngleProperty() { return startAngle; }

    /** When true we start placing slices clockwise from the startAngle */
    private BooleanProperty clockwise = new StyleableBooleanProperty(true) {
        @Override public void invalidated() {
            get();
            requestChartLayout();
        }

        @Override
        public Object getBean() {
            return PieChart.this;
        }

        @Override
        public String getName() {
            return "clockwise";
        }
```

```
278
279             public CssMetaData<PieChart,Boolean> getCssMetaData() {
280                 return StyleableProperties.CLOCKWISE;
281             }
282         };
283         public final void setClockwise(boolean value) { clockwise.setValue(value);}
284         public final boolean isClockwise() { return clockwise.getValue(); }
285         public final BooleanProperty clockwiseProperty() { return clockwise; }
286
287
288         /** The length of the line from the outside of the pie to the slice labels. */
289         private DoubleProperty labelLineLength = new StyleableDoubleProperty(20d) {
290             @Override public void invalidated() {
291                 get();
292                 requestChartLayout();
293             }
294
295             @Override
296             public Object getBean() {
297                 return PieChart.this;
298             }
299
300             @Override
301             public String getName() {
302                 return "labelLineLength";
303             }
304
305             public CssMetaData<PieChart,Number> getCssMetaData() {
306                 return StyleableProperties.LABEL_LINE_LENGTH;
307             }
308         };
309         public final double getLabelLineLength() { return labelLineLength.getValue(); }
310         public final void setLabelLineLength(double value) { labelLineLength.setValue(
        value); }
311         public final DoubleProperty labelLineLengthProperty() { return labelLineLength; }
312
313         /** When true pie slice labels are drawn */
314         private BooleanProperty labelsVisible = new StyleableBooleanProperty(true) {
315             @Override public void invalidated() {
316                 get();
317                 requestChartLayout();
318             }
319
320             @Override
321             public Object getBean() {
322                 return PieChart.this;
323             }
324
325             @Override
326             public String getName() {
327                 return "labelsVisible";
328             }
329
330             public CssMetaData<PieChart,Boolean> getCssMetaData() {
331                 return StyleableProperties.LABELS_VISIBLE;
332             }
333         };
334         public final void setLabelsVisible(boolean value) { labelsVisible.setValue(value
        );}
335
336         /**
337          * Indicates whether pie slice labels are drawn or not
338          * @return true if pie slice labels are visible and false otherwise.
339          */
340         public final boolean getLabelsVisible() { return labelsVisible.getValue(); }
341         public final BooleanProperty labelsVisibleProperty() { return labelsVisible; }
342
343         // -------------- CONSTRUCTOR ----------------------------------------------
344
345         /**
346          * Construct a new empty PieChart.
```

```
347         */
348        public PieChart() {
349            this(FXCollections.<Data>observableArrayList());
350        }
351
352        /**
353         * Construct a new PieChart with the given data
354         *
355         * @param data The data to use, this is the actual list used so any changes to
            it will be reflected in the chart
356         */
357        public PieChart(ObservableList<PieChart.Data> data) {
358            getChartChildren().add(labelLinePath);
359            labelLinePath.getStyleClass().add("chart-pie-label-line");
360            setLegend(legend);
361            setData(data);
362            // set chart content mirroring to be always false i.e. chartContent
                mirrorring is not done
363            // when  node orientation is right-to-left for PieChart.
364            useChartContentMirroring = false;
365        }
366
367        // -------------- METHODS ------------------------------------------------
368
369        private void dataNameChanged(Data item) {
370            item.textNode.setText(item.getName());
371            requestChartLayout();
372            updateLegend();
373        }
374
375        private void dataPieValueChanged(Data item) {
376            if (shouldAnimate()) {
377                animate(
378                    new KeyFrame(Duration.ZERO, new KeyValue(item.currentPieValueProperty
                    (),
379                            item.getCurrentPieValue())),
380                    new KeyFrame(Duration.millis(500),new KeyValue(item.
                    currentPieValueProperty(),
381                            item.getPieValue(), Interpolator.EASE_BOTH))
382                );
383            } else {
384                item.setCurrentPieValue(item.getPieValue());
385                requestChartLayout(); // RT-23091
386            }
387        }
388
389        private Node createArcRegion(Data item) {
390            Node arcRegion = item.getNode();
391            // check if symbol has already been created
392            if (arcRegion == null) {
393                arcRegion = new Region();
394                arcRegion.setNodeOrientation(NodeOrientation.LEFT_TO_RIGHT);
395                arcRegion.setPickOnBounds(false);
396                item.setNode(arcRegion);
397            }
398            return arcRegion;
399        }
400
401        private Text createPieLabel(Data item) {
402            Text text = item.textNode;
403            text.setText(item.getName());
404            return text;
405        }
406
407        private void updateDataItemStyleClass(final Data item, int index) {
408            Node node = item.getNode();
409            if (node != null) {
410                // Note: not sure if we want to add or check, ie be more careful and
                    efficient here
411                node.getStyleClass().setAll("chart-pie", "data" + index,
412                        "default-color" + item.defaultColorIndex % 8);
```

```
413                 if (item.getPieValue() < 0) {
414                     node.getStyleClass().add("negative");
415                 }
416             }
417         }
418 
419         private void dataItemAdded(final Data item, int index) {
420             // create shape
421             Node shape = createArcRegion(item);
422             final Text text = createPieLabel(item);
423             item.getChart().getChartChildren().add(shape);
424             if (shouldAnimate()) {
425                 // if the same data item is being removed, first stop the remove
                    animation,
426                 // remove the item and then start the add animation.
427                 if (dataRemoveTimeline != null && dataRemoveTimeline.getStatus().equals(
                    Animation.Status.RUNNING)) {
428                     if (dataItemBeingRemoved == item) {
429                         dataRemoveTimeline.stop();
430                         dataRemoveTimeline = null;
431                         getChartChildren().remove(item.textNode);
432                         getChartChildren().remove(shape);
433                         removeDataItemRef(item);
434                     }
435                 }
436                 animate(
437                     new KeyFrame(Duration.ZERO,
438                         new KeyValue(item.currentPieValueProperty(), item.
                            getCurrentPieValue()),
439                         new KeyValue(item.radiusMultiplierProperty(), item.
                            getRadiusMultiplier())),
440                     new KeyFrame(Duration.millis(500),
441                             actionEvent -> {
442                                 text.setOpacity(0);
443                                 // RT-23597 : item's chart might have been set to null if
444                                 // this item is added and removed before its add
                                    animation finishes.
445                                 if (item.getChart() == null) item.setChart(PieChart.this);
446                                 item.getChart().getChartChildren().add(text);
447                                 FadeTransition ft = new FadeTransition(Duration.millis(
                                    150),text);
448                                 ft.setToValue(1);
449                                 ft.play();
450                             },
451                         new KeyValue(item.currentPieValueProperty(), item.getPieValue(),
                            Interpolator.EASE_BOTH),
452                         new KeyValue(item.radiusMultiplierProperty(), 1, Interpolator.
                            EASE_BOTH))
453                 );
454             } else {
455                 getChartChildren().add(text);
456                 item.setRadiusMultiplier(1);
457                 item.setCurrentPieValue(item.getPieValue());
458             }
459 
460             // we sort the text nodes to always be at the end of the children list, so
                they have a higher z-order
461             // (Fix for RT-34564)
462             for (int i = 0; i < getChartChildren().size(); i++) {
463                 Node n = getChartChildren().get(i);
464                 if (n instanceof Text) {
465                     n.toFront();
466                 }
467             }
468         }
469 
470         private void removeDataItemRef(Data item) {
471             if (begin == item) {
472                 begin = item.next;
473             } else {
474                 Data ptr = begin;
```

```
475                 while(ptr != null && ptr.next != item) {
476                     ptr = ptr.next;
477                 }
478                 if(ptr != null) ptr.next = item.next;
479             }
480         }
481 
482         private Timeline createDataRemoveTimeline(final Data item) {
483             final Node shape = item.getNode();
484             Timeline t = new Timeline();
485             t.getKeyFrames().addAll(new KeyFrame(Duration.ZERO,
486                         new KeyValue(item.currentPieValueProperty(), item.
                            getCurrentPieValue()),
487                         new KeyValue(item.radiusMultiplierProperty(), item.
                            getRadiusMultiplier())),
488                     new KeyFrame(Duration.millis(500),
489                             actionEvent -> {
490                                 // removing item
491                                 colorBits.clear(item.defaultColorIndex);
492                                 getChartChildren().remove(shape);
493                                 // fade out label
494                                 FadeTransition ft = new FadeTransition(Duration.millis(
                                    150),item.textNode);
495                                 ft.setFromValue(1);
496                                 ft.setToValue(0);
497                                 ft.setOnFinished(new EventHandler<ActionEvent>() {
498                                     @Override public void handle(ActionEvent actionEvent
                                        ) {
499                                         getChartChildren().remove(item.textNode);
500                                         // remove chart references from old data -
                                            RT-22553
501                                         item.setChart(null);
502                                         removeDataItemRef(item);
503                                         item.textNode.setOpacity(1.0);
504                                     }
505                                 });
506                                 ft.play();
507                             },
508                         new KeyValue(item.currentPieValueProperty(), 0, Interpolator.
                            EASE_BOTH),
509                         new KeyValue(item.radiusMultiplierProperty(), 0))
510                     );
511             return t;
512         }
513 
514         private void dataItemRemoved(final Data item) {
515             final Node shape = item.getNode();
516             if (shouldAnimate()) {
517                 dataRemoveTimeline = createDataRemoveTimeline(item);
518                 dataItemBeingRemoved = item;
519                 animate(dataRemoveTimeline);
520             } else {
521                 colorBits.clear(item.defaultColorIndex);
522                 getChartChildren().remove(item.textNode);
523                 getChartChildren().remove(shape);
524                 // remove chart references from old data
525                 item.setChart(null);
526                 removeDataItemRef(item);
527             }
528         }
529 
530         /** @inheritDoc */
531         @Override protected void layoutChartChildren(double top, double left, double
            contentWidth, double contentHeight) {
532             centerX = contentWidth/2 + left;
533             centerY = contentHeight/2 + top;
534             double total = 0.0;
535             for (Data item = begin; item != null; item = item.next) {
536                 total+= Math.abs(item.getCurrentPieValue());
537             }
538             double scale = (total != 0) ? 360 / total : 0;
```

```
539
540             labelLinePath.getElements().clear();
541              // calculate combined bounds of all labels & pie radius
542             double[] labelsX = null;
543             double[] labelsY = null;
544             double[] labelAngles = null;
545             double labelScale = 1;
546             ArrayList<LabelLayoutInfo> fullPie = null;
547             boolean shouldShowLabels = getLabelsVisible();
548             if(getLabelsVisible()) {
549
550                 double xPad = 0d;
551                 double yPad = 0d;
552
553                 labelsX = new double[getDataSize()];
554                 labelsY = new double[getDataSize()];
555                 labelAngles = new double[getDataSize()];
556                 fullPie = new ArrayList<LabelLayoutInfo>();
557                 int index = 0;
558                 double start = getStartAngle();
559                 for (Data item = begin; item != null; item = item.next) {
560                     // remove any scale on the text node
561                     item.textNode.getTransforms().clear();
562
563                     double size = (isClockwise()) ? (-scale * Math.abs(item.
                        getCurrentPieValue())) : (scale * Math.abs(item.getCurrentPieValue
                        ()));
564                     labelAngles[index] = normalizeAngle(start + (size / 2));
565                     final double sproutX = calcX(labelAngles[index], getLabelLineLength
                        (), 0);
566                     final double sproutY = calcY(labelAngles[index], getLabelLineLength
                        (), 0);
567                     labelsX[index] = sproutX;
568                     labelsY[index] = sproutY;
569                     xPad = Math.max(xPad, 2 * (item.textNode.getLayoutBounds().getWidth()
                         + LABEL_TICK_GAP + Math.abs(sproutX)));
570                     if (sproutY > 0) { // on bottom
571                         yPad = Math.max(yPad, 2 * Math.abs(sproutY+item.textNode.
                            getLayoutBounds().getMaxY()));
572                     } else { // on top
573                         yPad = Math.max(yPad, 2 * Math.abs(sproutY + item.textNode.
                            getLayoutBounds().getMinY()));
574                     }
575                     start+= size;
576                     index++;
577                 }
578                 pieRadius = Math.min(contentWidth - xPad, contentHeight - yPad) / 2;
579                 // check if this makes the pie too small
580                 if (pieRadius < MIN_PIE_RADIUS ) {
581                     // calculate scale for text to fit labels in
582                     final double roomX = contentWidth-MIN_PIE_RADIUS-MIN_PIE_RADIUS;
583                     final double roomY = contentHeight-MIN_PIE_RADIUS-MIN_PIE_RADIUS;
584                     labelScale = Math.min(
585                             roomX/xPad,
586                             roomY/yPad
587                     );
588                     // hide labels if pie radius is less than minimum
589                     if ((begin == null && labelScale < 0.7) || ((begin.textNode.getFont
                        ().getSize()*labelScale) < 9)) {
590                         shouldShowLabels = false;
591                         labelScale = 1;
592                     } else {
593                         // set pieRadius to minimum
594                         pieRadius = MIN_PIE_RADIUS;
595                         // apply scale to all label positions
596                         for(int i=0; i< labelsX.length; i++) {
597                             labelsX[i] =  labelsX[i] * labelScale;
598                             labelsY[i] =  labelsY[i] * labelScale;
599                         }
600                     }
601                 }
```

```
602         }
603
604         if(!shouldShowLabels) {
605             pieRadius = Math.min(contentWidth,contentHeight) / 2;
606         }
607
608         if (getChartChildren().size() > 0) {
609             int index = 0;
610             for (Data item = begin; item != null; item = item.next) {
611                 // layout labels for pie slice
612                 item.textNode.setVisible(shouldShowLabels);
613                 if (shouldShowLabels) {
614                     double size = (isClockwise()) ? (-scale * Math.abs(item.
                        getCurrentPieValue())) : (scale * Math.abs(item.
                        getCurrentPieValue()));
615                     final boolean isLeftSide = !(labelAngles[index] > -90 &&
                        labelAngles[index] < 90);
616
617                     double sliceCenterEdgeX = calcX(labelAngles[index], pieRadius,
                        centerX);
618                     double sliceCenterEdgeY = calcY(labelAngles[index], pieRadius,
                        centerY);
619                     double xval = isLeftSide ?
620                         (labelsX[index] + sliceCenterEdgeX - item.textNode.
                        getLayoutBounds().getMaxX() - LABEL_TICK_GAP) :
621                         (labelsX[index] + sliceCenterEdgeX - item.textNode.
                        getLayoutBounds().getMinX() + LABEL_TICK_GAP);
622                     double yval = labelsY[index] + sliceCenterEdgeY - (item.textNode.
                        getLayoutBounds().getMinY()/2) -2;
623
624                     // do the line (Path)for labels
625                     double lineEndX = sliceCenterEdgeX +labelsX[index];
626                     double lineEndY = sliceCenterEdgeY +labelsY[index];
627                     LabelLayoutInfo info = new LabelLayoutInfo(sliceCenterEdgeX,
628                             sliceCenterEdgeY,lineEndX, lineEndY, xval, yval, item.
                            textNode, Math.abs(size));
629                     fullPie.add(info);
630
631                     // set label scales
632                     if (labelScale < 1) {
633                         item.textNode.getTransforms().add(
634                             new Scale(
635                                     labelScale, labelScale,
636                                     isLeftSide ? item.textNode.getLayoutBounds().
                                    getWidth() : 0,
637 //                                    0,
638                                     0
639                             )
640                         );
641                     }
642                 }
643                 index++;
644             }
645
646              // Check for collision and resolve by hiding the label of the smaller
                pie slice
647             resolveCollision(fullPie);
648
649             // update/draw pie slices
650             double sAngle = getStartAngle();
651             for (Data item = begin; item != null; item = item.next) {
652              Node node = item.getNode();
653                 Arc arc = null;
654                  if (node != null) {
655                     if (node instanceof Region) {
656                         Region arcRegion = (Region)node;
657                         if( arcRegion.getShape() == null) {
658                             arc = new Arc();
659                             arcRegion.setShape(arc);
660                         } else {
661                             arc = (Arc)arcRegion.getShape();
```

```
                        }
                        arcRegion.setShape(null);
                        arcRegion.setShape(arc);
                        arcRegion.setScaleShape(false);
                        arcRegion.setCenterShape(false);
                        arcRegion.setCacheShape(false);
                    }
                }
                double size = (isClockwise()) ? (-scale * Math.abs(item.
                getCurrentPieValue())) : (scale * Math.abs(item.getCurrentPieValue
                ()));
                // update slice arc size
                arc.setStartAngle(sAngle);
                arc.setLength(size);
                arc.setType(ArcType.ROUND);
                arc.setRadiusX(pieRadius * item.getRadiusMultiplier());
                arc.setRadiusY(pieRadius * item.getRadiusMultiplier());
                node.setLayoutX(centerX);
                node.setLayoutY(centerY);
                sAngle += size;
            }
            // finally draw the text and line
            if (fullPie != null) {
                for (LabelLayoutInfo info : fullPie) {
                    if (info.text.isVisible()) drawLabelLinePath(info);
                }
            }
        }
    }

    // We check for pie slice label collision and if collision is detected, we then
    // compare the size of the slices, and hide the label of the smaller slice.
    private void resolveCollision(ArrayList<LabelLayoutInfo> list) {
        int boxH = (begin != null) ? (int)begin.textNode.getLayoutBounds().getHeight
        () : 0;
        int i; int j;
        for (i = 0, j = 1; list != null && j < list.size(); j++ ) {
            LabelLayoutInfo box1 = list.get(i);
            LabelLayoutInfo box2 = list.get(j);
            if ((box1.text.isVisible() && box2.text.isVisible()) &&
                    (fuzzyGT(box2.textY, box1.textY) ? fuzzyLT((box2.textY - boxH -
                    box1.textY), 2) :
                     fuzzyLT((box1.textY - boxH - box2.textY), 2)) &&
                    (fuzzyGT(box1.textX, box2.textX) ? fuzzyLT((box1.textX - box2.
                    textX), box2.text.prefWidth(-1)) :
                        fuzzyLT((box2.textX - box1.textX), box1.text.prefWidth(-1))))
                         {
                if (fuzzyLT(box1.size, box2.size)) {
                    box1.text.setVisible(false);
                    i = j;
                } else {
                    box2.text.setVisible(false);
                }
            } else {
                i = j;
            }
        }
    }

    private int fuzzyCompare(double o1, double o2) {
       double fuzz = 0.00001;
       return (((Math.abs(o1 - o2)) < fuzz) ? 0 : ((o1 < o2) ? -1 : 1));
    }

    private boolean fuzzyGT(double o1, double o2) {
        return (fuzzyCompare(o1, o2) == 1) ? true: false;
    }

    private boolean fuzzyLT(double o1, double o2) {
        return (fuzzyCompare(o1, o2) == -1) ? true : false;
    }
```

```
727
728         private void drawLabelLinePath(LabelLayoutInfo info) {
729             info.text.setLayoutX(info.textX);
730             info.text.setLayoutY(info.textY);
731             labelLinePath.getElements().add(new MoveTo(info.startX, info.startY));
732             labelLinePath.getElements().add(new LineTo(info.endX, info.endY));
733 
734             labelLinePath.getElements().add(new MoveTo(info.endX-LABEL_BALL_RADIUS,info.
                endY));
735             labelLinePath.getElements().add(new ArcTo(LABEL_BALL_RADIUS,
                LABEL_BALL_RADIUS,
736                         90, info.endX,info.endY-LABEL_BALL_RADIUS, false, true));
737             labelLinePath.getElements().add(new ArcTo(LABEL_BALL_RADIUS,
                LABEL_BALL_RADIUS,
738                         90, info.endX+LABEL_BALL_RADIUS,info.endY, false, true));
739             labelLinePath.getElements().add(new ArcTo(LABEL_BALL_RADIUS,
                LABEL_BALL_RADIUS,
740                         90, info.endX,info.endY+LABEL_BALL_RADIUS, false, true));
741             labelLinePath.getElements().add(new ArcTo(LABEL_BALL_RADIUS,
                LABEL_BALL_RADIUS,
742                         90, info.endX-LABEL_BALL_RADIUS,info.endY, false, true));
743             labelLinePath.getElements().add(new ClosePath());
744         }
745         /**
746          * This is called whenever a series is added or removed and the legend needs to
                be updated
747          */
748         private void updateLegend() {
749             Node legendNode = getLegend();
750             if (legendNode != null && legendNode != legend) return; // RT-23569 dont
                update when user has set legend.
751             legend.setVertical(getLegendSide().equals(Side.LEFT) || getLegendSide().
                equals(Side.RIGHT));
752             legend.getItems().clear();
753             if (getData() != null) {
754                 for (Data item : getData()) {
755                     LegendItem legenditem = new LegendItem(item.getName());
756                     legenditem.getSymbol().getStyleClass().addAll(item.getNode().
                        getStyleClass());
757                     legenditem.getSymbol().getStyleClass().add("pie-legend-symbol");
758                     legend.getItems().add(legenditem);
759                 }
760             }
761             if (legend.getItems().size() > 0) {
762                 if (legendNode == null) {
763                     setLegend(legend);
764                 }
765             } else {
766                 setLegend(null);
767             }
768         }
769 
770         private int getDataSize() {
771             int count = 0;
772             for (Data d = begin; d != null; d = d.next) {
773                 count++;
774             }
775             return count;
776         }
777 
778         private static double calcX(double angle, double radius, double centerX) {
779             return (double)(centerX + radius * Math.cos(Math.toRadians(-angle)));
780         }
781 
782         private static double calcY(double angle, double radius, double centerY) {
783             return (double)(centerY + radius * Math.sin(Math.toRadians(-angle)));
784         }
785 
786          /** Normalize any angle into -180 to 180 deg range */
787         private static double normalizeAngle(double angle) {
788             double a = angle % 360;
```

```
789             if (a <= -180) a += 360;
790             if (a > 180) a -= 360;
791             return a;
792         }
793 
794         // -------------- INNER CLASSES ----------------------------------------------
795 
796         // Class holding label line layout info for collision detection and removal
797         final static class LabelLayoutInfo {
798             double startX;
799             double startY;
800             double endX;
801             double endY;
802             double textX;
803             double textY;
804             Text text;
805             double size;
806 
807             public LabelLayoutInfo(double startX, double startY, double endX, double endY,
808                     double textX, double textY, Text text, double size) {
809                 this.startX = startX;
810                 this.startY = startY;
811                 this.endX = endX;
812                 this.endY = endY;
813                 this.textX = textX;
814                 this.textY = textY;
815                 this.text = text;
816                 this.size = size;
817             }
818         }
819 
820         /**
821          * PieChart Data Item, represents one slice in the PieChart
822          *
823          * @since JavaFX 2.0
824          */
825         public final static class Data {
826 
827             private Text textNode = new Text();
828             /**
829              * Next pointer for the next data item : so we can do animation on data
                delete.
830              */
831             private Data next = null;
832 
833             /**
834              * Default color index for this slice.
835              */
836             private int defaultColorIndex;
837 
838             // -------------- PUBLIC PROPERTIES ----------------------------------
839 
840             /**
841              * The chart which this data belongs to.
842              */
843             private ReadOnlyObjectWrapper<PieChart> chart = new ReadOnlyObjectWrapper<
                PieChart>(this, "chart");
844 
845             public final PieChart getChart() {
846                 return chart.getValue();
847             }
848 
849             private void setChart(PieChart value) {
850                 chart.setValue(value);
851             }
852 
853             public final ReadOnlyObjectProperty<PieChart> chartProperty() {
854                 return chart.getReadOnlyProperty();
855             }
856 
857             /**
```

```
858             * The name of the pie slice
859             */
860            private StringProperty name = new StringPropertyBase() {
861                @Override
862                protected void invalidated() {
863                    if (getChart() != null) getChart().dataNameChanged(Data.this);
864                }
865
866                @Override
867                public Object getBean() {
868                    return Data.this;
869                }
870
871                @Override
872                public String getName() {
873                    return "name";
874                }
875            };
876
877            public final void setName(java.lang.String value) {
878                name.setValue(value);
879            }
880
881            public final java.lang.String getName() {
882                return name.getValue();
883            }
884
885            public final StringProperty nameProperty() {
886                return name;
887            }
888
889            /**
890             * The value of the pie slice
891             */
892            private DoubleProperty pieValue = new DoublePropertyBase() {
893                @Override
894                protected void invalidated() {
895                    if (getChart() != null) getChart().dataPieValueChanged(Data.this);
896                }
897
898                @Override
899                public Object getBean() {
900                    return Data.this;
901                }
902
903                @Override
904                public String getName() {
905                    return "pieValue";
906                }
907            };
908
909            public final double getPieValue() {
910                return pieValue.getValue();
911            }
912
913            public final void setPieValue(double value) {
914                pieValue.setValue(value);
915            }
916
917            public final DoubleProperty pieValueProperty() {
918                return pieValue;
919            }
920
921            /**
922             * The current pie value, used during animation. This will be the last data
                 value, new data value or
923             * anywhere in between
924             */
925            private DoubleProperty currentPieValue = new SimpleDoubleProperty(this,
                "currentPieValue");
926
```

```
927             private double getCurrentPieValue() {
928                 return currentPieValue.getValue();
929             }
930
931             private void setCurrentPieValue(double value) {
932                 currentPieValue.setValue(value);
933             }
934
935             private DoubleProperty currentPieValueProperty() {
936                 return currentPieValue;
937             }
938
939             /**
940              * Multiplier that is used to animate the radius of the pie slice
941              */
942             private DoubleProperty radiusMultiplier = new SimpleDoubleProperty(this,
                "radiusMultiplier");
943
944             private double getRadiusMultiplier() {
945                 return radiusMultiplier.getValue();
946             }
947
948             private void setRadiusMultiplier(double value) {
949                 radiusMultiplier.setValue(value);
950             }
951
952             private DoubleProperty radiusMultiplierProperty() {
953                 return radiusMultiplier;
954             }
955
956             /**
957              * Readonly access to the node that represents the pie slice. You can use
                this to add mouse event listeners etc.
958              */
959             private ReadOnlyObjectWrapper<Node> node = new ReadOnlyObjectWrapper<>(this,
                "node");
960
961             /**
962              * Returns the node that represents the pie slice. You can use this to
963              * add mouse event listeners etc.
964              */
965             public Node getNode() {
966                 return node.getValue();
967             }
968
969             private void setNode(Node value) {
970                 node.setValue(value);
971             }
972
973             public ReadOnlyObjectProperty<Node> nodeProperty() {
974                 return node.getReadOnlyProperty();
975             }
976
977             // -------------- CONSTRUCTOR
                ------------------------------------------------
978
979             /**
980              * Constructs a PieChart.Data object with the given name and value.
981              *
982              * @param name  name for Pie
983              * @param value pie value
984              */
985             public Data(java.lang.String name, double value) {
986                 setName(name);
987                 setPieValue(value);
988                 textNode.getStyleClass().addAll("text", "chart-pie-label");
989                 textNode.setAccessibleRole(AccessibleRole.TEXT);
990                 textNode.setAccessibleRoleDescription("slice");
991                 textNode.focusTraversableProperty().bind(Platform.
                    accessibilityActiveProperty());
992                 textNode.accessibleTextProperty().bind( new StringBinding() {
```

```
                {bind(nameProperty(), currentPieValueProperty());}
                @Override protected String computeValue() {
                    return getName() + " represents " + getCurrentPieValue() + "
                    percent";
                }
            });
        }

        // -------------- PUBLIC METHODS
        ----------------------------------------------

        /**
         * Returns a string representation of this {@code Data} object.
         *
         * @return a string representation of this {@code Data} object.
         */
        @Override
        public java.lang.String toString() {
            return "Data[" + getName() + "," + getPieValue() + "]";
        }
    }

    // -------------- STYLESHEET HANDLING -----------------------------------

    /**
      * Super-lazy instantiation pattern from Bill Pugh.
      * @treatAsPrivate implementation detail
      */
     private static class StyleableProperties {
         private static final CssMetaData<PieChart,Boolean> CLOCKWISE =
             new CssMetaData<PieChart,Boolean>("-fx-clockwise",
                 BooleanConverter.getInstance(), Boolean.TRUE) {

            @Override
            public boolean isSettable(PieChart node) {
                return node.clockwise == null || !node.clockwise.isBound();
            }

            @Override
            public StyleableProperty<Boolean> getStyleableProperty(PieChart node) {
                return (StyleableProperty<Boolean>)(WritableValue<Boolean>)node.
                clockwiseProperty();
            }
        };

         private static final CssMetaData<PieChart,Boolean> LABELS_VISIBLE =
             new CssMetaData<PieChart,Boolean>("-fx-pie-label-visible",
                 BooleanConverter.getInstance(), Boolean.TRUE) {

            @Override
            public boolean isSettable(PieChart node) {
                return node.labelsVisible == null || !node.labelsVisible.isBound();
            }

            @Override
            public StyleableProperty<Boolean> getStyleableProperty(PieChart node) {
                return (StyleableProperty<Boolean>)(WritableValue<Boolean>)node.
                labelsVisibleProperty();
            }
        };

         private static final CssMetaData<PieChart,Number> LABEL_LINE_LENGTH =
             new CssMetaData<PieChart,Number>("-fx-label-line-length",
                 SizeConverter.getInstance(), 20d) {

            @Override
            public boolean isSettable(PieChart node) {
                return node.labelLineLength == null || !node.labelLineLength.isBound
                ();
            }
```

```
1059                @Override
1060                public StyleableProperty<Number> getStyleableProperty(PieChart node) {
1061                    return (StyleableProperty<Number>)(WritableValue<Number>)node.
                        labelLineLengthProperty();
1062                }
1063            };
1064
1065             private static final CssMetaData<PieChart,Number> START_ANGLE =
1066                 new CssMetaData<PieChart,Number>("-fx-start-angle",
1067                     SizeConverter.getInstance(), 0d) {
1068
1069                @Override
1070                public boolean isSettable(PieChart node) {
1071                    return node.startAngle == null || !node.startAngle.isBound();
1072                }
1073
1074                @Override
1075                public StyleableProperty<Number> getStyleableProperty(PieChart node) {
1076                    return (StyleableProperty<Number>)(WritableValue<Number>)node.
                        startAngleProperty();
1077                }
1078            };
1079
1080             private static final List<CssMetaData<? extends Styleable, ?>> STYLEABLES;
1081             static {
1082
1083                final List<CssMetaData<? extends Styleable, ?>> styleables =
1084                    new ArrayList<CssMetaData<? extends Styleable, ?>>(Chart.
                        getClassCssMetaData());
1085                styleables.add(CLOCKWISE);
1086                styleables.add(LABELS_VISIBLE);
1087                styleables.add(LABEL_LINE_LENGTH);
1088                styleables.add(START_ANGLE);
1089                STYLEABLES = Collections.unmodifiableList(styleables);
1090             }
1091        }
1092
1093        /**
1094         * @return The CssMetaData associated with this class, which may include the
1095         * CssMetaData of its super classes.
1096         * @since JavaFX 8.0
1097         */
1098        public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
1099            return StyleableProperties.STYLEABLES;
1100        }
1101
1102        /**
1103         * {@inheritDoc}
1104         * @since JavaFX 8.0
1105         */
1106        @Override
1107        public List<CssMetaData<? extends Styleable, ?>> getCssMetaData() {
1108            return getClassCssMetaData();
1109        }
1110
1111    }
1112
1113
1114
```