Custom renderers sample

This example covers:

- **A custom renderer for the package.** The package without inner elements is filled with a green color.
- **A custom renderer for the slot.** The slot values are rounded.
- **A custom renderer for the dependency link.** The dependency link is a blue thicker line with custom line ends.

This example shows, how to create custom renderers for the preceding pointed symbols.

The custom package renderer. The empty package (that is without inner elements) is filled with green color:

```
    class PackageRenderer extends ShapeRenderer
    {
        public Color getColor(PresentationElement presentationElement,
PresentationElementColorEnum colorEnum)
        {
            if (PresentationElementColorEnum.FILL.equals(colorEnum))
            {
                // the color to fill
                Element element = presentationElement.getElement();
                if (element instanceof com.nomagic.uml2.ext.magicdraw.
classes.mdkernel.Package &&
                    !((com.nomagic.uml2.ext.magicdraw.classes.mdkernel.
Package) element).hasOwnedElement())
                {
                    // the package has no elements, use the green color to
fill
                    return Color.GREEN;
                }
            }
            return super.getColor(presentationElement, colorEnum);
        }
    }
```

The custom slot renderer (used in the instance specification symbol). The slot values are rounded:

```
    class SlotRenderer extends ShapeRenderer
    {
        public String getText(PresentationElement presentationElement,
PresentationElementTextEnum textEnum)
        {
            if (PresentationElementTextEnum.NAME.equals(textEnum))
            {
                // the slot text is shown as a name
                Element element = presentationElement.getElement();
                if (element instanceof Slot)
                {
                    Slot slot = (Slot) element;
                    if (slot.hasValue())
                    {
                        String string = "";
                        List<ValueSpecification> values = slot.getValue();
                        for (ValueSpecification value : values)
                        {
                            if (value instanceof LiteralString)
                            {
                                LiteralString literalString =
(LiteralString) value;
                                if (string.length() > 0)
                                {
                                   string += "; ";

                                }
                                String literalValue = literalString.
getValue();
                                try
                                {
                                    // round a value
                                    double doubleValue = Double.parseDouble
(literalValue);
                                    double rounded = Math.round
(doubleValue * 100) / 100;
                                    literalValue = Double.toString
(rounded);
                                }
                                catch (NumberFormatException e)
                                {
                                }
                                string += literalValue;
                            }
                        }
                        return slot.getDefiningFeature().getName() + "=" +
string;
                    }
                }
            }
            return super.getText(presentationElement, textEnum);
        }
    }
```

The custom dependency link renderer. The dependency link is a blue thicker line with custom line ends:

```
    class DependencyRenderer extends PathRenderer
    {
        private PathEndRenderer mClientEndRenderer;
        DependencyRenderer()
        {
            // a custom client end renderer, use a filled circle at the end
            mClientEndRenderer = new PathEndRenderer(PathEndAdornment.
CIRCLE, PathEndAdornmentModifier.FILLED);
        }

        public Color getColor(PresentationElement presentationElement,
PresentationElementColorEnum colorEnum)
        {
            if (PresentationElementColorEnum.LINE.equals(colorEnum))
            {
                // use a blue color for a line
                return Color.BLUE;
            }
            return super.getColor(presentationElement, colorEnum);
        }


        protected PathEndRenderer getClientEndRenderer(PathElement
pathElement)
        {
            // use a custom end renderer
            return mClientEndRenderer;
        }

        public int getLineWidth(PresentationElement presentationElement)
        {
            // a line width is 2
            return 2;
        }

        protected void drawPathAdornment(Graphics g, PathElement
pathElement)
        {
            super.drawPathAdornment(g, pathElement);

            // draw a circle at the middle of the dependency line
            Color background = Color.WHITE;
            Property property = pathElement.
getDiagramPresentationElement().getProperty(PropertyID.
DIAGRAM_BACKGROUND_COLOR);
            if (property != null)
            {
                Object value = property.getValue();
                if (value instanceof Color)
                {
                    background = (Color) value;
                }
            }
            Point middlePoint = pathElement.getMiddlePoint();
            int diameter = 10;
            int radius = diameter / 2;
            int x = middlePoint.x - radius;
            int y = middlePoint.y - radius;
            Color color = g.getColor();
            g.setColor(background);
            g.fillOval(x, y, diameter, diameter);
            g.setColor(color);
            g.drawOval(x, y, diameter, diameter);
        }
    }
```