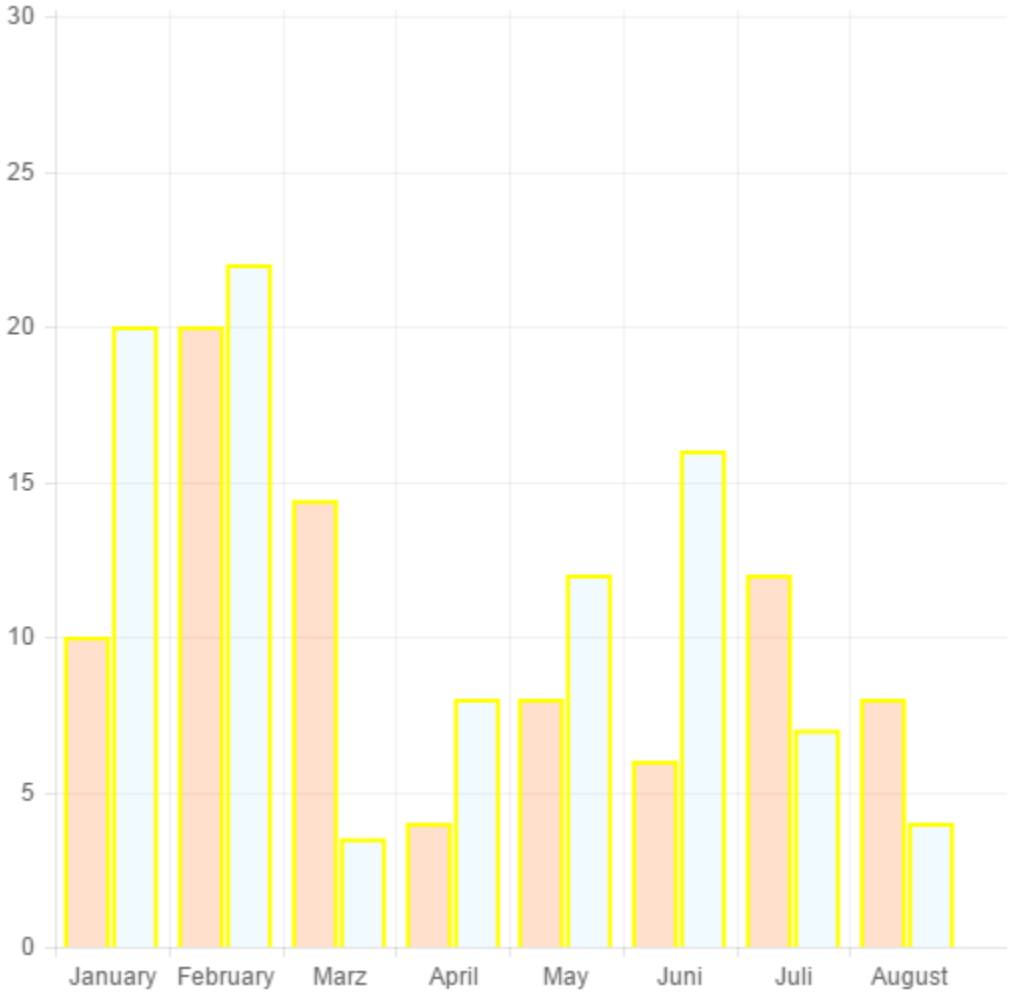
Chart Documentation

1. Chart Types

1.1 Bar Chart



a) Creating new instance of the bar chart

```
new BarChart(container, nameOfTheHtmlElement, dataModelForTheChart);
```

- **container –** the parent java container where the object should be embedded

- **nameOfTheHtmlElement –** name of the html element (div) which will display the chart. If you want to embed this chart in special template you will have to call in the vtl template **$control.get(nameOfTheHtmlElement)** to display the chart in your new template. If you don't need to create special template, enough will be to create the instance in the java class like presented above.
- **dataModelForTheChart –** data model containing information which will be presented in the chart. In case of **BarChart** we need to use **ValueListChartModel** , which contains the list of labels (presented on X) and list of **ValueListChartDataset** which contains the list of datasets. Each of the dataset contains the list of of values (should be in the same size as the list of labels, and gui settings in what way the **BarChart** should present delivered data (color, size, background, font) other settings for the chart can be found under **barChart.getConfiguration()** method and can be overwritten every time.

b) Assigning data to the bar chart

Data assignment can be done in two different ways.

- Assigning, updating an removing using methods delivered in ChartModel

  1) **addDataToModel** – adding defined element in given dataset number under given label
  2) **changeDataByModel** – change the value of the element defined by dataset which it is located in and given label
  3) **removeDataFromModel** – removes element from the dataset number defined and label given in the method

- Changing the whole model using the method **setModel** accessible in the chart instance.

  Both ways don't require refreshing the view as it is already done in the implementation.

c) Defining available events for bar chart

At the time there is only one type of event accessible – **actionClick**. In order to add your own implementation for the event it is required to create a class implementing **SelectionListener** and add to the chart using method **chart. addElementSelectedListener** . You can register as many listeners as you want.

d) Bar Chart Configuration

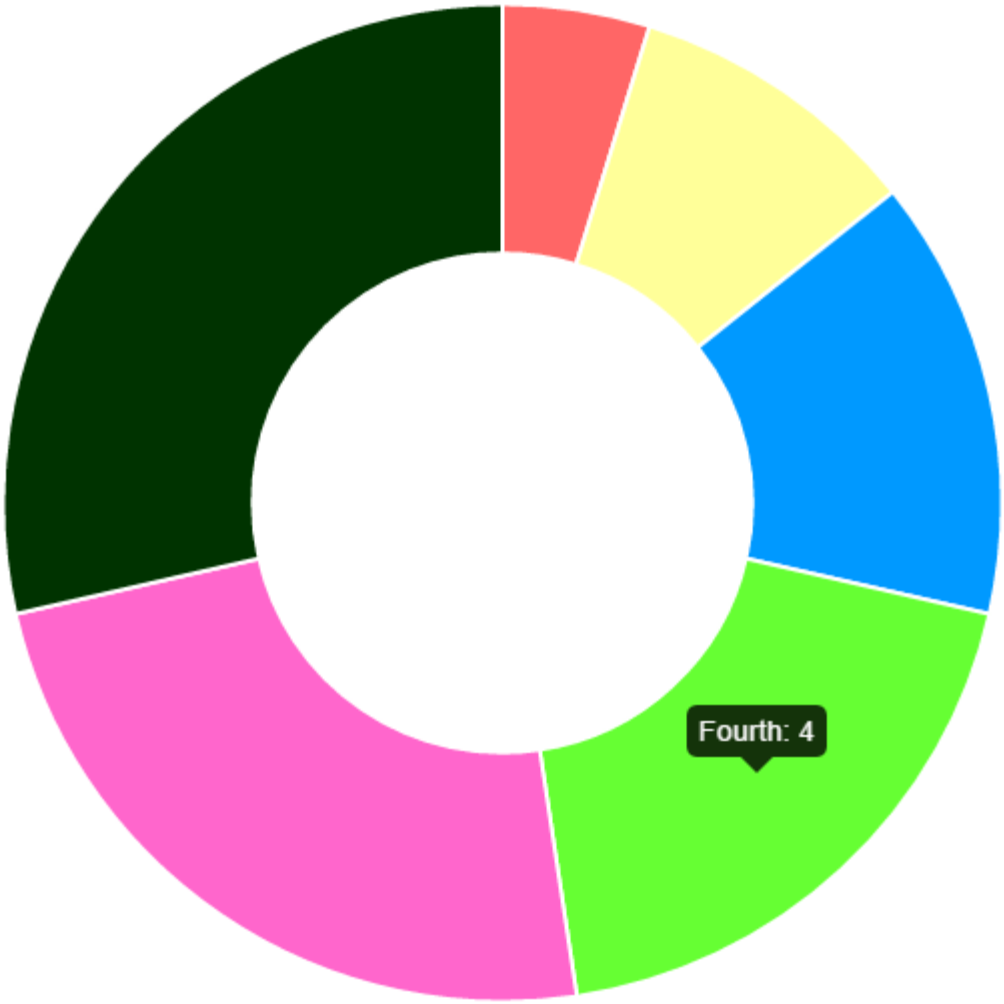
- **barValueSpacing** - Whether the scale should start at zero, or // an order of magnitude down from the lowest value – value in pixels
- **barDatasetSpacing** = Spacing between each of the X value sets. Value in pixels

The BarChart provides default configuration which could be changed by

- using the method chart.getConfiguration().set…
- providing new instance of BarChartConfiguration

Additional settings provided in the **BarChartConfiguration** come from the common configuration and are accessible and common for all chart types

1.2 Circle Chart



a) Creating new instance of the circle chart

```
new CircleChart(container, nameOfTheHtmlElement, dataModelForTheChart);
```

- **container –** the parent java container where the object should be embedded
- **nameOfTheHtmlElement –** name of the html element (div) which will display the chart. If you want to embed this chart in special template you will have to call in the vtl template **$control.get(nameOfTheHtmlElement)** to display the chart in your new template. If you don't need to create special template, enough will be to create the instance in the java class like presented above.
- **dataModelForTheChart –** data model containing information which will be presented in the chart. In case of **CircleChart** we need to use **SimpleValueChartModel** only a list of **SimpleValueChartDataset** which

contains definition for each value with the label presented in the chart. Behind defining the label and value user needs also to define the color of the segment and the highlight color of the segment for each dataset defined. The difference between **SimpleValueChartDataset** and **ValueListChartDataset** is that each dataset contains only one value and only one value can be assigned to one label.

b) Assigning data to the circle chart

Data assignment can be done in two different ways.

- Assigning, updating an removing using methods delivered in **SimpleValueChartModel**

  1) **addDataToModel** – adding given value with color for the displaying under given new label
  2) **changeDataByModel** – change the value of the element in given dataset
  3) **removeDataFromModel** – removes dataset defined by given label

- Changing the whole model using the method **setModel** accessible in the chart instance.

  Both ways don't require refreshing the view as it is already done in the implementation.

c) Defining available events for circle chart

At the time there is only one type of event accessible – **actionClick**. In order to add your own implementation for the event it is required to create a class implementing **SelectionListener** and add to the chart using method **chart. addElementSelectedListener** . You can register as many listeners as you want.

d) Circle Chart Configuration

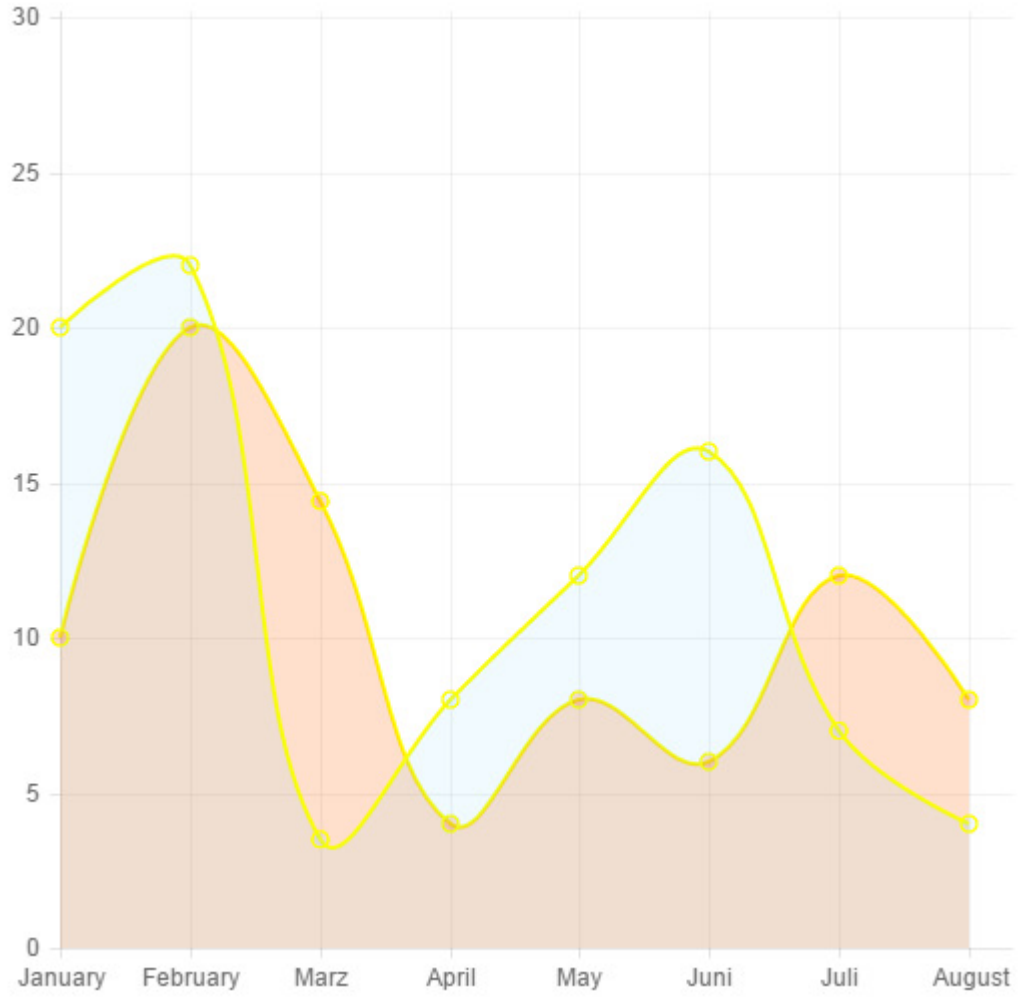
- **percentageInnerCutout**- Number - The percentage of the chart that we cut out of the

The BarChart provides default configuration which could be changed by

- Using the method **chart.getConfiguration().set…**
- providing new instance of **CircleChartConfiguration**

Additional settings provided in the **BarChartConfiguration** come from the common configuration and are accessible and common for all chart types

1.3 Line Chart



a) Creating new instance of the line chart

```
new LineChart(container, nameOfTheHtmlElement, dataModelForTheChart);
```

- **container –** the parent java container where the object should be embedded

- **nameOfTheHtmlElement –** name of the html element (div) which will display the chart. If you want to embed this chart in special template you will have to call in the vtl template **$control.get(nameOfTheHtmlElement)** to display the chart in your new template. If you don't need to create special template, enough will be to create the instance in the java class like presented above.
- **dataModelForTheChart –** data model containing information which will be presented in the chart. In case of **LineChart** we need to use **ValueListChartModel** , which contains the list of labels (presented on X) and list of **ValueListChartDataset** which contains the list of datasets. Each of the dataset contains the list of of values (should be in the same size as the list of labels, and gui settings in what way the **LineChart** should present delivered data (color, size, background, font) other settings for the chart can be found under **lineChart.getConfiguration()** method and can be overwritten every time.

b) Assigning data to the line chart

Data assignment can be done in two different ways.

- Assigning, updating an removing using methods delivered in ChartModel

c) **addDataToModel** – adding defined element in given dataset number under given label
d) **changeDataByModel** – change the value of the element defined by dataset which it is located in and given label
e) **removeDataFromModel** – removes element from the dataset number defined and label given in the method

- Changing the whole model using the method **setModel** accessible in the chart instance.

  Both ways don't require refreshing the view as it is already done in the implementation.

c) Defining available events for line chart

At the time there is only one type of event accessible – **actionClick**. In order to add your own implementation for the event it is required to create a class implementing **SelectionListener** and add to the chart using method **chart. addElementSelectedListener** . You can register as many listeners as you want.

d) Line Chart Configuration

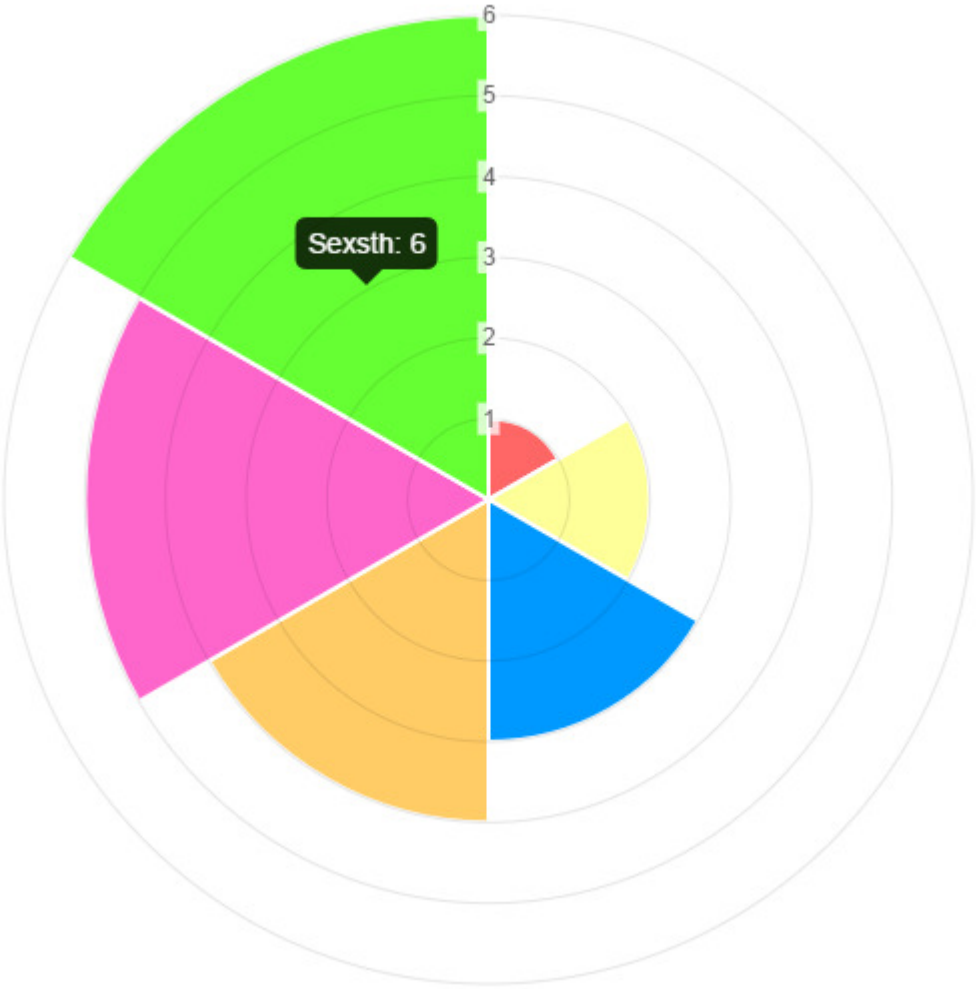
- **bezierCurveTension** - Number - Tension of the bezier curve between points
- **pointDot** - Boolean - Whether to show a dot for each point
- **bezierCurve** - - Boolean - Whether the line is curved between points
- **pointDotRadius** - Number - Radius of each point dot in pixels
- **pointDotStrokeWidth** - Number - Pixel width of point dot stroke
- **pointHitDetectionRadius** - Number - amount extra to add to the radius to cater for hit detection outside the drawn point
- **datasetFill** - Boolean - Whether to fill the dataset with a colour

The LineChart provides default configuration which could be changed by

- using the method **chart.getConfiguration().set…**
- providing new instance of **LineChartConfiguration**

Additional settings provided in the **LineChartConfiguration** come from the common configuration and are accessible and common for all chart types

1.4 Polar Chart



a) Creating new instance of the polar chart

```
new PolarChart(container, nameOfTheHtmlElement, dataModelForTheChart);
```

- **container –** the parent java container where the object should be embedded
- **nameOfTheHtmlElement –** name of the html element (div) which will display the chart. If you want to embed this chart in special template you will have to call in the vtl template **$control.get(nameOfTheHtmlElement)** to display the chart in your new template. If you don't need to create special template, enough will be to create the instance in the java class like presented above.
- **dataModelForTheChart –** data model containing information which will be presented in the chart. In case of **PolarChart** we need to use **SimpleValueChartModel** only a list of **SimpleValueChartDataset** which contains definition for each value with the label presented in the chart. Behind defining the label and value user needs also to define the color of the segment

and the highlight color of the segment for each dataset defined. The difference between **SimpleValueChartDataset** and **ValueListChartDataset** is that each dataset contains only one value and only one value can be assigned to one label.

b) Assigning data to the polar chart

Data assignment can be done in two different ways.

- Assigning, updating an removing using methods delivered in **SimpleValueChartModel**

  a) **addDataToModel** – adding given value with color under given new label
  b) **changeDataByModel** – change the value of the element in given dataset
  c) **removeDataFromModel** – removes dataset defined by given label

- Changing the whole model using the method **setModel** accessible in the chart instance.

  Both ways don't require refreshing the view as it is already done in the implementation.

c) Defining available events for bar chart

At the time there is only one type of event accessible – **actionClick**. In order to add your own implementation for the event it is required to create a class implementing **SelectionListener** and add to the chart using method **chart. addElementSelectedListener** . You can register as many listeners as you want.

d) Polar Chart Configuration

- **scaleShowLine**- Boolean - Show line for each value in the scale
- **scaleBackdropPaddingX** - Number - The backdrop padding to the side of the label in in pixels
- **scaleBackdropPaddingY** - Number - The backdrop padding above & below the label in in pixels
- **scaleBackdropColor** - String - The colour of the label backdrop
- **scaleShowLabelBackdrop** - Boolean - Show a backdrop to the scale label

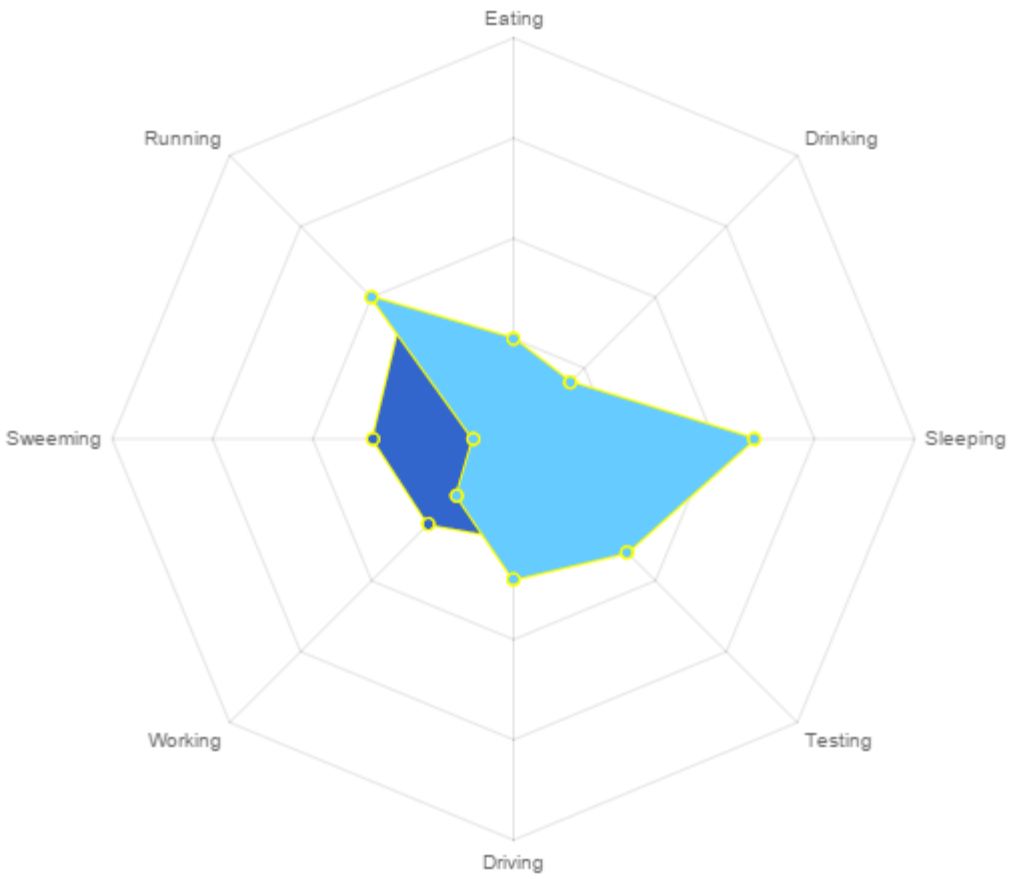
The PolarChart provides default configuration which could be changed by

- using the **method chart.getConfiguration().set…**

- providing new instance of **PolarChartConfiguration**

  Additional settings provided in the **PolarChartConfiguration** come from the common configuration and are accessible and common for all chart types

1.5 Radar Chart



a) Creating new instance of the radar chart

```
new RadarChart(container, nameOfTheHtmlElement, dataModelForTheChart);
```

- **container –** the parent java container where the object should be embedded
- **nameOfTheHtmlElement –** name of the html element (div) which will display the chart. If you want to embed this chart in special template you will have to call in the vtl template **$control.get(nameOfTheHtmlElement)** to display the

chart in your new template. If you don't need to create special template, enough will be to create the instance in the java class like presented above.

- **dataModelForTheChart –** data model containing information which will be presented in the chart. In case of **RadarChart** we need to use **RadarChartModel** which has the same structure as **ValueListChartModel** , and contains the list of labels (presented on X) and list of **RadarChartDataset** (the same structure as ValueListDataset except two additional gui settings which can be provided by displaying the chart) **.** Each of the dataset contains the list of of values (should be in the same size as the list of labels, and gui settings in what way the **RadarChart** should present delivered data (color, size, background, font) other settings for the chart can be found under **radarChart.getConfiguration()** method and can be overwritten every time.

b) Assigning data to the radar chart

Data assignment can be done in two different ways.

- Assigning, updating an removing using methods delivered in ChartModel

c) **addDataToModel** – adding defined element in given dataset number under given label
d) **changeDataByModel** – change the value of the element defined by dataset which it is located in and given label
e) **removeDataFromModel** – removes element from the dataset number defined and label given in the method

- Changing the whole model using the method **setModel** accessible in the chart instance.

Both ways don't require refreshing the view as it is already done in the implementation.

c) Defining available events for radar chart

At the time there is only one type of event accessible – **actionClick**. In order to add your own implementation for the event it is required to create a class implementing **SelectionListener** and add to the chart using method **chart. addElementSelectedListener** . You can register as many listeners as you want.

d) Radar Chart Configuration

- **scaleShowLabelBackdrop**- Show a backdrop to the scale label
- **scaleBackdropColor**- The colour of the label backdrop
- **scaleBackdropPaddingY** Number - The backdrop padding above & below the label in pixels
- **scaleBackdropPaddingX** - Number - The backdrop padding to the side of the label in pixels
- **scaleShowLine** - Boolean - Show line for each value in the scale

The RadarChart provides default configuration which could be changed by

- using the method chart.getConfiguration().set…
- providing new instance of RadarChartConfiguration

Additional settings provided in the RadarChartConfiguration come from the common configuration and are accessible and common for all chart types

2. Common Chart Configuration

Accessible in all chart format types, provided with default values which can be changed each time by using the method **chart.getChartConfiguration().set…(setter method)**

All settings provided for each chart are listed below :

a) **Width** - in pixels height of the chart, only taken under account if chart responsive is set to false. As default is set : 500
b) **Height** - in pixels height of the chart, only taken under account if chart responsive is set to false. As default is set : 500
c) **Responsive** - Boolean - whether or not the chart should be responsive and resize when the browser does. As default is set : false
d) **Animation** - Boolean - Whether to animate the chart. As default is set : true
e) **animationSteps** - Number - Number of animation steps. As default is set : 60
f) **animationEasing** - Animation easing effect. Defined as enumeration. As default is set : ***EASEINOUTBOUNCE***
g) **showScale** - Boolean - If we should show the scale at all. As default is set : true
h) **scaleOverride** - Boolean - If we want to override with a hard coded scale. As default is set : false
i) **scaleLineColor** - String - Color of the scale line. As default is set : `rgba(0,0,0,1)`
j) **scaleLineWidth** - Number - Pixel width of the scale line. As default is set : 1

k) **scaleShowLabels** - Boolean - Whether to show labels on the scale. As default is set : true
l) **scaleBeginAtZero** - Whether the scale should start at zero, or an order. As default is set : true
m) **scaleFontFamily** - Scale label font declaration for the scale label. As default is set : `Helvetica Neue', 'Helvetica'; 'Arial', sans-serif`
n) **scaleFontSize** - Number - Scale label font size in pixels. As default is set : 12
o) **scaleFontStyle** - Number - Scale label font weight style. As default is set : `normal`
p) **scaleFontColor** - String - Scale label font colour. As default is set : `#666`
q) **maintainAspectRatio** - Boolean - whether to maintain the starting aspect ratio or not when responsive, if set to false, will take up entire container. As default is set : true
r) **showTooltips** - Boolean - Determines whether to draw tooltips on the canvas or or not. As default is set : true
s) **tooltipFillColor** - String - Tooltip background colour. As default is set : `rgba(0,0,0,0.8)`
t) **tooltipFontFamily** - String - Tooltip label font declaration for the scale label. As default is set : `Helvetica Neue', 'Helvetica', 'Arial', sans-serif`

u) **tooltipFontSize**- Number - Tooltip label font size in pixels. As default is set : 14
v) **tooltipFontStyle** - String - Tooltip font weight style. As default is set : `normal`
w) **tooltipFontColor** - String - Tooltip label font colour. As default is set : `#fff`
x) **tooltipTitleFontFamily** - String - Tooltip title font declaration for the scale label. As default is set : `'Helvetica Neue', 'Helvetica'; 'Arial', sans-serif`
y) **tooltipTitleFontSize** - Number - Tooltip title font size in pixels. As default is set : 14
z) **tooltipTitleFontStyle** - String - Tooltip title font weight style. As default is set : `bold`
aa) **tooltipTitleFontColor** - String - Tooltip title font weight style. As default is set : `#fff`
bb) **tooltipYPadding** - Number - pixel height of padding around tooltip text. As default is set : 6
cc) **tooltipXPadding** - Number - pixel width of padding around tooltip text. As default is set : 6
dd) **tooltipCaretSize** - Number - Size of the caret on the tooltip. As default is set : 8
ee) **tooltipCornerRadius** - Number - Pixel radius of the tooltip border. As default is set : 6
ff) tooltipXOffset - Number - Pixel offset from point x to tooltip edge. As default is set : 10
gg) **tooltipTemplate** - String - Template string for single tooltips. As default is set : `<%if (label){%><%=label%>= <%}%><%= value %>`
hh) **legendTemplate** - A legend template, format for the legend tooltip. For each of the chart different
ii) **showStroke** - Whether we should show a stroke on each segment. As default is set : true
jj) **strokeColor** - The colour of each segment stroke. As default is set : `rgba(0,0,0,1)`

kk) **segmentStrokeWidth** - Number - The width of each segment stroke. As default is set : 2

ll) **scaleGridLineColor** - Color of the grid lines. As default is set : `rgba(0,0,0,1)`

mm) **scaleGridLineWidth** - Number - Width of the grid lines. As default is set : 1

nn) **scaleShowVerticalLines** - Boolean - Whether to show vertical lines (except Y axis). As default is set : false

oo) **scaleShowHorizontalLines** - Whether to show horizontal lines (except X axis). As default is set : false