Version 3.1

OFC-Charts, Manuel Utilisateur

Oxymel S.A.

Le Campus
6, rue Jean-Pierre Timbaud
78180 Montigny le Bretonneux

FRANCE

Type du Document	*Manuel Utilisateur*
Référence	OFC-Charts
Date	Juin 2006

Sommaire

1. Les composants Oxymel

L'architecture des systèmes d'informations modernes est généralement basée sur la notion d'architecture n-tiers. De telles architectures permettent de séparer clairement les différents niveaux de traitements informatiques nécessaire pour la mise en œuvre d'une application.

Classiquement au nombre de trois, ces tiers tendent à résoudre les problématiques suivantes :

- La présentation de l'information à l'utilisateur ;
- La résolution des problèmes relatifs au métier de l'utilisateur ;
- La persistance des données.

Dans ce contexte, l'offre logicielle d'Oxymel, baptisée OFC pour "**Oxymel Foundation Classes**" regroupe un ensemble de composants dédiés à l'intégration et à la publication de données sur Internet. La vocation des OFC est de faciliter la collaboration entre les développeurs Java et les autres intervenants impliqués dans un projet de base de données en ligne, et de permettre un multi-accès à l'information (assistants personnels, téléphones mobiles, ordinateurs).

Facilitant l'intégration à l'existant, OFC Connection répond au besoin d'accès à une base de données pour intégrer l'information dans un applicatif Java notamment pour une mise en ligne sur Internet.

- **OFC Connection** prend en charge la connectivité à la base de données, il permet de personnaliser le couplage objet-relationnel et automatise la génération des requêtes SQL de recherche, de mise à jour et de destruction.

Composants de publication, OFC Publishing, OFC Reporting et OFC Charts répondent au besoin de publier de l'information sur des supports complémentaires Internet tels que HTML, XML, WML ou sur un support papier notamment par un format texte ou pdf.

- **OFC Publishing** permet l'intégration de l'information dans des modèles de publication Internet tel que HTML.
- **OFC Reporting** permet la génération dynamique de documents dans les formats binaires notamment PDF.
- **OFC Charts** permet la génération dynamique de graphiques qui pourront être intégrés dans les pages HTML ou les document PDF.

2. OFC-Charts: Présentation

OFC Charts est un produit 100% *JAVA* qui offre une API simple pour la génération de graphiques. Ces graphiques générés dans différents formats, peuvent être intégrés dans des pages WEB ou dans des rapports PDF.

2.1 Format des images

OFC-Charts permet de générer des images sous différents formats :

- JPEG : appel de la méthode `drawJPEG( … )`
- PNG : appel de la méthode `drawPNG( … )`
- SVG : appel de la méthode `drawSVG( … )`
- SVG animé : appel de la méthode `drawAnimateSVG( … )`

2.2 Types de graphiques

OFC-Charts permet de construire différents types de graphiques, ce sont principalement des :

- Courbes (**VerticalLinePlot**)
- Histogrammes représentés graphiquement de plusieurs manières :
  - Vertical ou horizontal (**VerticalBarChart**)
  - Sous forme de courbes de distribution (**VerticalLineChart**)
  - Empilée (**VerticalStackedBarChart**)
  - Combinée avec un ou deux axes des ordonnées (**CombinedVerticalBarChartXY , CombinedVerticalBarChartXYY**). Dans le cas des graphiques combinés, plusieurs graphiques sont superposés, les représentations peuvent être différentes d'un graphique à l'autre (courbe de distribution ou histogramme).
- Camemberts ou secteurs (**PieChart, ExplodedPieChart**)
- Courbes d'iso valeur (**Contour2D**). Ce type de courbe permet une représentation graphique en 2 dimensions d'une relation entre trois données. Deux d'entre elles sont utilisées sur les axes x et y et la troisième est utilisée pour les contours. Ces contours sont représentés par des lignes, les zones entre les courbes sont colorées pour représenter l'interpolation des valeurs.

La représentation des graphiques s'effectue à partir de plusieurs informations comme la taille du graphique en pixels (hauteur et largeur), des informations sur les axes (horizontal et vertical) et bien entendu les données associées.

OFC-Charts génère automatiquement chaque graphique dans le format demandé (png, jpeg ou svg) en tenant compte de la taille souhaitée.

Les graphiques contenant un ensemble unique de séries peuvent être animés dans un fichier SVG. L'animation consiste en un affichage progressif des différentes séries.

2.3 Attributs de présentation

OFC-Charts propose un éventail important d'attributs ou propriétés qui vont permettre d'agir sur l'apparence du graphique. Les principales sont indiquées ci-dessous :

- Taille de l'image et positionnement du graphique dans cette zone ;

- Couleur de fond ;
- Personnalisation de la palette de couleurs ;
- Personnalisation de la légende : position, style de police, couleur et écriture sur une ou plusieurs lignes. La légende peut être positionnée ou non ;
- Possibilité de visualiser une grille ou non, les caractéristiques des traits sont paramétrables (couleur, taille…) ;
- Les axes peuvent être présents ou non sur le graphique, ils peuvent être gradués ou non. L'utilisateur peut choisir l'épaisseur du trait, sa couleur ainsi que la couleur des graduations. Les axes peuvent se terminer par une flèche ou non et posséder un label dont la couleur et la police sont paramétrables. Chaque axe possède sa propre échelle qui peut être fixée par l'utilisateur ou calculée automatiquement par le système ;
- La position des axes, dont les labels sont des nombres, est paramétrable. L'utilisateur peut décaler une valeur afin de rendre les graphiques plus lisibles. Sur les axes horizontaux, l'utilisateur peut spécifier un angle d'inclinaison pour l'affichage des labels.
- Pour les courbes, matérialisation des points par un symbole prédéfini ;
- Présentation en 3D des histogrammes et camemberts, espacement des catégories, choix des de la profondeur des barres en 3D.

Ce document contient un ensemble d'exemples permettant de mieux comprendre le fonctionnement d'OFC-Charts. Par souci pédagogique, ces exemples sont le plus simple possible et utilisent la présentation par défaut intégrée dans les classes de génération de graphique. La documentation complète des interfaces, classes et méthodes se trouve dans le javadoc livré avec le composant.

3. Vertical Line Plot

3.1 Description

La classe ***VerticalLinePlot*** permet la génération de courbes définies par une succession de point x,y.

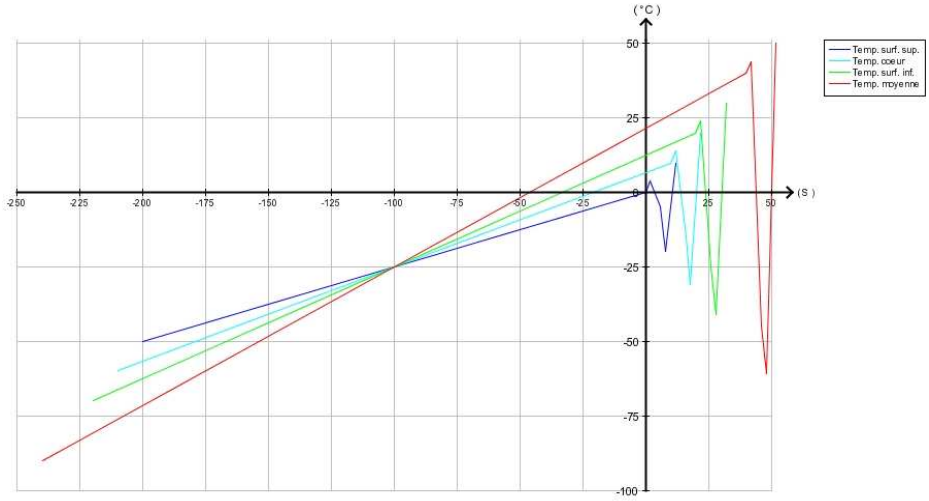
Les informations relatives à une courbe sont regroupées dans une entité appelée *série*. Une série est définie par un identifiant et contient :

- un intitulé : il correspond à l'information caractéristique de la série et est utilisé dans la légende ,
- une succession de points : dans le cas des courbes, il s'agit de couples définissant les coordonnées x et y des points.

Un graphique peut contenir plusieurs courbes, chacune d'entre elle étant définie par une série. L'ensemble des séries est regroupé dans une entité nommée *container*.

3.2 Exemple

Dans l'exemple suivant quatre courbes sont présentées, elles correspondent à quatre séries.



3.3 Code de l'exemple

Chaque courbe est définie par une succession de points. Les informations relatives à chaque série de points, c'est-à-dire l'ensemble des coordonnées x,y des points mais aussi l'intitulé de cette série dans la légende, sont contenues dans la classe *Point2DSeries*.

La classe *Point2DSeriesContainer* contient l'ensemble des séries représentées dans le graphique, c'est-à-dire l'ensemble des Point2DSeries.

La classe *VerticalLinePlot* permet de dessiner le graphique de la taille (hauteur, largeur) souhaitée. Pour cela, il faut lui donner les informations sur les axes X et Y, ainsi que l'ensemble des séries définissant les courbes. Ce dernier est donné par l'intermédiaire d'un container de séries.

```
// Création du conteneur de courbes
Point2DSeriesContainer point2DSeriesContainer = new Point2DSeriesContainer();

// Courbe 1  (construction de la série – paramètres : identifiant et légende - et ajout des  points)
Point2DSeries point2DSeries = new Point2DSeries("S1", "Temp. surf. sup.");
point2DSeries.addPoint2D(new Point2D(-200, -50));
point2DSeries.addPoint2D(new Point2D(0, 0));
point2DSeries.addPoint2D(new Point2D(2, 4));
point2DSeries.addPoint2D(new Point2D(6, -5));
point2DSeries.addPoint2D(new Point2D(8, -20));
point2DSeries.addPoint2D(new Point2D(12, 10));

// Ajout de la courbe dans le conteneur
point2DSeriesContainer.addPoint2DSeries(point2DSeries);

// Courbe 2  (construction de la série et ajout des points)
point2DSeries = new Point2DSeries("S2", "Temp. coeur");
point2DSeries.addPoint2D(new Point2D(-210, -60));
point2DSeries.addPoint2D(new Point2D(10, 10));
point2DSeries.addPoint2D(new Point2D(12, 14));
point2DSeries.addPoint2D(new Point2D(16, -15));
point2DSeries.addPoint2D(new Point2D(18, -31));
point2DSeries.addPoint2D(new Point2D(22, 20));

// Ajout de la courbe dans le conteneur
point2DSeriesContainer.addPoint2DSeries(point2DSeries);

// Courbe 3  (construction de la série et ajout des points)
point2DSeries = new Point2DSeries("S3", "Temp. surf. inf.");
point2DSeries.addPoint2D(new Point2D(-220, -70));
point2DSeries.addPoint2D(new Point2D(20, 20));
point2DSeries.addPoint2D(new Point2D(22, 24));
point2DSeries.addPoint2D(new Point2D(26, -25));
point2DSeries.addPoint2D(new Point2D(28, -41));
point2DSeries.addPoint2D(new Point2D(32, 30));

// Ajout de la courbe dans le conteneur
point2DSeriesContainer.addPoint2DSeries(point2DSeries);

// Courbe 4
point2DSeries = new Point2DSeries("S4", "Temp. moyenne");
point2DSeries.addPoint2D(new Point2D(-240, -90));
point2DSeries.addPoint2D(new Point2D(40, 40));
point2DSeries.addPoint2D(new Point2D(42, 44));
point2DSeries.addPoint2D(new Point2D(46, -45));
point2DSeries.addPoint2D(new Point2D(48, -61));
point2DSeries.addPoint2D(new Point2D(52, 50));

// Ajout de la courbe dans le conteneur
point2DSeriesContainer.addPoint2DSeries(point2DSeries);

// Génération du graphique au format JPEG
// Taille de l'image
int imageHeight = 500; int imageWidth = 900;
// En paramètre de la construction du graphique les axes avec leur légende
VerticalLinePlot verticalLinePlot = new VerticalLinePlot(imageWidth, imageHeight, new HorizontalNumberAxis("(   S )"),
new VerticalNumberAxis( "( °C )"), point2DSeriesContainer);
// Génération effective de l'image au format jpeg
  verticalLinePlot.drawJPEG("test.jpg");
```

4. Vertical Bar Chart

4.1 Description

La classe ***VerticalBarChart*** permet la génération d'histogrammes. Cette classe permet de traiter plusieurs séries de données numériques et de produire le graphique correspondant sous forme d'histogramme.

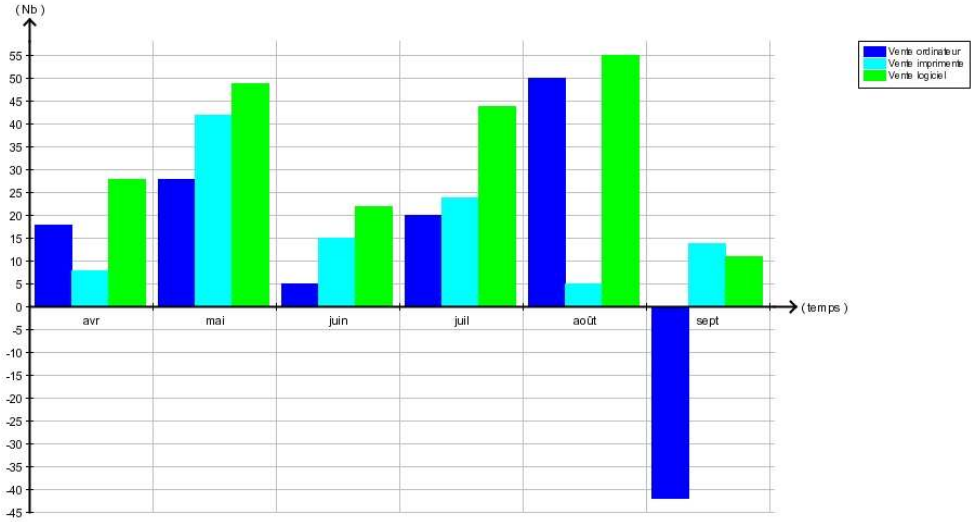
Un histogramme permet une partition en catégories des différentes valeurs contenues dans les séries. Chaque série contenant des valeurs relatives aux différentes catégories, *VerticalBarChart* n'impose pas de restriction sur les données numériques en entrée.

Une série est caractérisée par un identifiant et contient un intitulé permettant de construire la légende et une série d'information qui sont des couples contenant une catégorie et une valeur numérique positive ou négative.

L'ensemble de ces données, définitions de catégories et de séries, est regroupé dans un container.

4.2 Exemple

L'exemple ci-dessous est une représentation mensuelle de la différence entre le chiffre d'affaire réel et les prévisions de vente de matériel, ordinateurs et imprimantes, et de logiciel. Les trois séries représentent respectivement la vente d'ordinateurs, d'imprimantes et de logiciels sur une période de 6 mois (avril à septembre).



4.3 Code de l'exemple

La classe *CategoryValueSeriesContainer* contient toutes les séries ainsi que la définition de l'ensemble des catégories de l'histogramme.

La classe *CategorieValueSerie* contient toutes les données relatives à une série, c'est-à-dire l'ensemble des valeurs pour chaque catégorie, mais aussi l'intitulé de cette série dans la légende.

La classe *VerticalBarChart* permet de dessiner le graphique de la taille (hauteur, largeur) souhaitée. Pour cela, il faut lui donner les informations sur les axes X et Y, ainsi que l'ensemble des séries et catégories par l'intermédiaire d'un objet de la classe *CategoryValueSeriesContainer*.

```
// Création du Conteneur de série
CategoryValueSeriesContainer categoryValueSeriesContainer = new CategoryValueSeriesContainer();

// Définition des catégories de l'histogramme
categoryValueSeriesContainer.addCategory(new Category("id1","avr"));    // Ajout categorie 1
categoryValueSeriesContainer.addCategory(new Category("id2","mai"));   // Ajout categorie 2
categoryValueSeriesContainer.addCategory(new Category("id3","juin"));  // Ajout categorie 3
categoryValueSeriesContainer.addCategory(new Category("id4","juil"));  // Ajout categorie 4
categoryValueSeriesContainer.addCategory(new Category("id5","août")); // Ajout categorie 5
categoryValueSeriesContainer.addCategory(new Category("id6","sept")); // Ajout categorie 6

// Série 1 (construction de la série et ajout des valeurs pour chaque catégorie)
CategoryValueSeries categoryValueSeries = new CategoryValueSeries("S1", "Vente ordinateur");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 18));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 20));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 50));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", -42));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 2 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S2", "Vente imprimante");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 8));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 42));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 15));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 24));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 14));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 3 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S3", "Vente logiciel");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 49));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 22));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 44));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 55));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 11));

// Ajout  de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Génération du graphique au format JPEG
int imageHeight = 500; int imageWidth = 900;
VerticalBarChart verticalBarChart = new VerticalBarChart(imageWidth, imageHeight, new HorizontalCategoryAxis("( temps )"), new
VerticalNumberAxis("( Nb )"), categoryValueSeriesContainer);
verticalBarChart.drawJPEG("test.jpg");
```

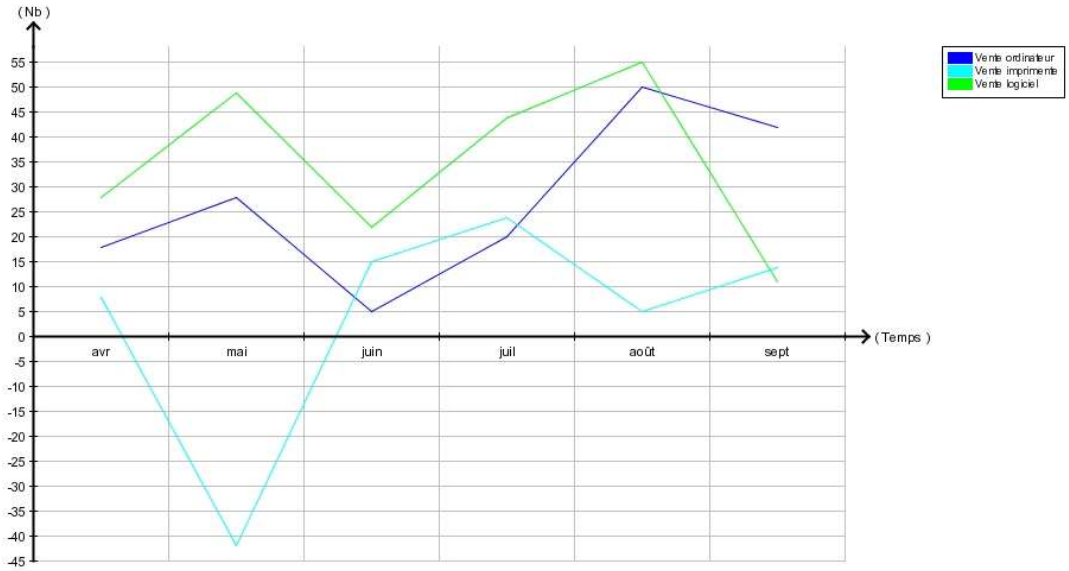
5. Vertical Line Chart

5.1 Description

La classe ***VerticalLineChart*** permet la génération de courbes de distribution. Ce type de graphique est équivalent aux histogrammes, une série est alors représentée par une courbe construite en reliant les points correspondants aux sommets des barres de l'histogramme. Les données permettant de construire une courbe de distribution sont identiques à celles qui permettent de construire un histogramme.

5.2 Exemple

Reprenons l'exemple du paragraphe précédent, qui permet une représentation graphique de la différence entre chiffre d'affaire effectué et chiffre d'affaire prévisionnel sur une période de 6 mois. Les catégories correspondent aux mois d'avril à septembre et trois séries sont présentées : une série pour la vente d'ordinateur, une série pour la vente d'imprimante et enfin une série pour la vente de logiciel.



5.3 Code de l'exemple

La classe *CategoryValueSeriesContainer* contient toutes les séries ainsi que la définition de l'ensemble des catégories représentées.

La classe *CategorieValueSerie* contient toutes les données relatives à une série c'est-à-dire l'ensemble des valeurs pour chaque catégorie, mais aussi l'intitulé de cette série dans la légende.

La classe *VerticalLineChart* permet de dessiner le graphique à la taille (hauteur, largeur) souhaitée. Pour cela, il faut lui donner les informations sur les axes X et Y, ainsi que l'ensemble des séries.

```
// Création du Conteneur de série
CategoryValueSeriesContainer categoryValueSeriesContainer = new CategoryValueSeriesContainer();

// Définition des catégories de l'histogramme
categoryValueSeriesContainer.addCategory(new Category("id1","avr")); // Ajout categorie 1
categoryValueSeriesContainer.addCategory(new Category("id2","mai")); // Ajout categorie 2
categoryValueSeriesContainer.addCategory(new Category("id3","juin")); // Ajout categorie 3
categoryValueSeriesContainer.addCategory(new Category("id4","juil")); // Ajout categorie 4
categoryValueSeriesContainer.addCategory(new Category("id5","août")); // Ajout categorie 5
categoryValueSeriesContainer.addCategory(new Category("id6","sept")); // Ajout categorie 6

// Série 1 (construction de la série et ajout des valeurs pour chaque catégorie)
CategoryValueSeries categoryValueSeries = new CategoryValueSeries("S1", "Vente ordinateur");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 18));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 20));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 50));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 42));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 2 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S2", "Vente imprimante");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 8));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", -42));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 15));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 24));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 14));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 3 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S3", "Vente logiciel");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 49));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 22));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 44));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 55));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 11));

// Ajout  de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Génération du graphique au format JPEG
int imageHeight = 500; int imageWidth = 900;
VerticalLineChart verticalLineChart = new VerticalLineChart(imageWidth, imageHeight, new HorizontalCategoryAxis("( temps )"), new
VerticalNumberAxis("( Nb )"), categoryValueSeriesContainer);
verticalLineChart.drawJPEG("test.jpg");
```

6. Vertical Stacked Bar Chart

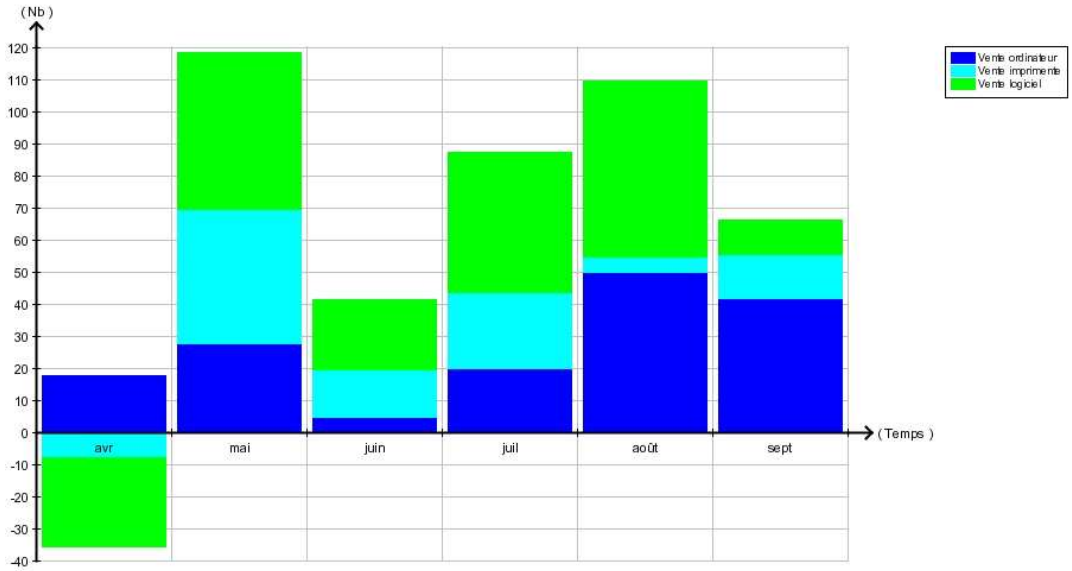
6.1 Description

La classe ***VerticalStackedBarChart*** permet la génération d'histogrammes empilés. D'un point de vue graphique, chaque catégorie est représentée par une barre construite par empilement des valeurs dans les différentes séries pour cette catégorie. Chaque série contient les valeurs relatives aux catégories représentées dans l'histogramme. *VerticalStackedBarChart* gère des valeurs positives et négatives, pour une catégorie donnée les valeurs positives sont empilées au-dessus de l'axe, les valeurs négatives sont empilées au-dessous de cet axe (voir exemple).

Les données permettant de construire un histogramme empilé sont identiques à celles qui permettent de construire un histogramme.

6.2 Exemple

Dans l'exemple ci-après, les catégories correspondent aux mois d'avril à septembre. Dans cet exemple, trois séries sont présentées : une série pour la vente d'ordinateurs, une série pour la vente d'imprimantes et enfin une série pour la vente de logiciel.



6.3 Code de l'exemple

La classe *CategoryValueSeriesContainer* contient toutes les séries ainsi que la définition de l'ensemble des catégories de l'histogramme.

La classe *CategorieValueSerie* contient toutes les données relatives à une série c'est-à-dire l'ensemble des valeurs pour chaque catégorie, ainsi que l'intitulé de cette série dans la légende.

La classe *VerticalStackedBarChart* permet de dessiner le graphique de la taille (hauteur, largeur) souhaitée. Pour cela, il faut lui donner les informations sur les axes X et Y, ainsi que l'ensemble des séries par l'intermédiaire d'un objet de la classe *CategoryValueSeriesContainer*.

```
// Création du Conteneur de série
CategoryValueSeriesContainer categoryValueSeriesContainer = new CategoryValueSeriesContainer();

// Définition des catégories de l'histogramme
categoryValueSeriesContainer.addCategory(new Category("id1","avr")); //Ajout categorie 1
categoryValueSeriesContainer.addCategory(new Category("id2","mai")); //Ajout categorie 2
categoryValueSeriesContainer.addCategory(new Category("id3","juin")); //Ajout categorie 3
categoryValueSeriesContainer.addCategory(new Category("id4","juil")); //Ajout categorie 4
categoryValueSeriesContainer.addCategory(new Category("id5","août")); //Ajout categorie 5
categoryValueSeriesContainer.addCategory(new Category("id6","sept")); //Ajout categorie 6

// Série 1 (construction de la série et ajout des valeurs pour chaque catégorie)
CategoryValueSeries categoryValueSeries = new CategoryValueSeries("S1", "Vente ordinateur");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 18));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 20));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 50));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 42));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 2 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S2", "Vente imprimante");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", -8));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 42));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 15));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 24));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 5));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 14));

// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

// Série 3 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S3", "Vente logiciel");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", -28));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 49));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 22));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 44));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 55));
categoryValueSeries.addCategoryValue(new CategoryValue("id6", 11));

// Ajout  de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);

//Génération du graphique au format JPEG
int imageHeight = 500; int imageWidth = 900;
VerticalStackedBarChart  verticalStackedBarChart = new VerticalStackedBarChart(imageWidth, imageHeight,
    new  HorizontalCategoryAxis("( temps )"), new VerticalNumberAxis("( Nb )"), categoryValueSeriesContainer);
VerticalStackedBarChart.drawJPEG("test.jpg");
```

7. Combined Vertical Bar Chart XYY

7.1 Description

La classe ***CombinedVerticalBarChartXYY*** permet la génération d'un graphique composé de :

- deux histogrammes superposés,
- d'une ou plusieurs lignes horizontales.

Les catégories des deux histogrammes doivent être identiques, chacun des deux histogrammes possède son axe des ordonnées propre. Les deux axes sont représentés sur le graphique, l'un à droite et l'autre à gauche.

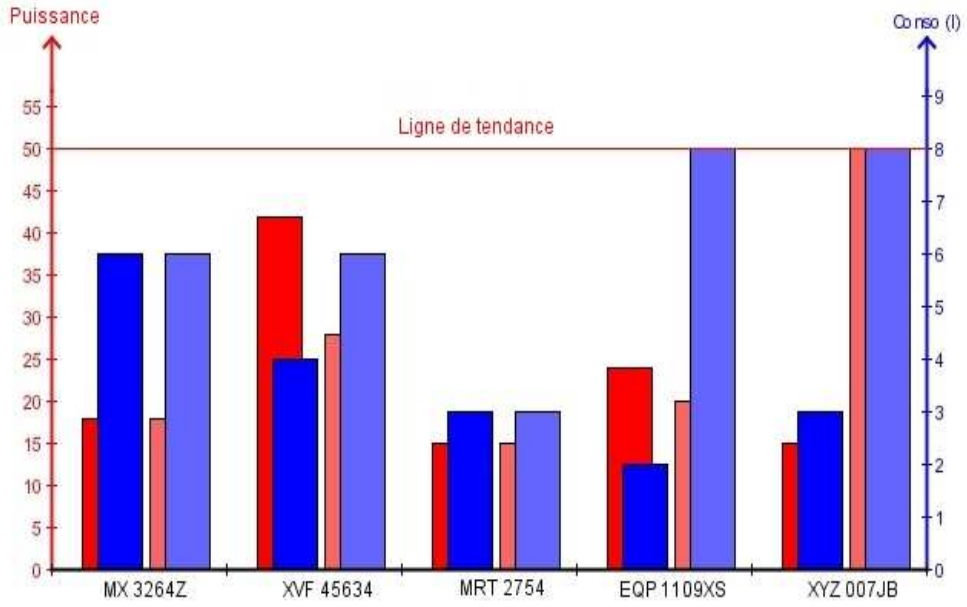
Les lignes horizontales sont identifiées par une valeur d'ordonnée et un axe, elles possèdent un intitulé.

Comme précédemment les données caractéristiques d'un histogramme sont stockées dans un container, sachant que les deux containers correspondant aux deux histogrammes doivent partager la même définition de catégories.

Une notion de renderer est introduite, elle permet de définir la position de l'axe (gauche ou droite) d'un histogramme ainsi que le décalage entre les barres, ceci afin qu'elles soient toutes visibles.

7.2 Exemple

Dans l'exemple ci-après deux histogrammes superposés sont représentés. Chacun d'eux contient deux séries de valeurs correspondant aux mêmes catégories. Cependant un histogramme (le rouge) est lié à l'axe des Y de gauche, alors que l'autre (le bleu) est lié à l'axe des Y de droite.



Une ligne horizontale d'information a été placée à la valeur 50 sur l'axe des Y de gauche.

7.3 Code de l'exemple

La classe *CategoryValueSeriesContainer* contient toutes les séries ainsi que la définition de l'ensemble des catégories d'un histogramme.

La classe *CategorieValueSerie* contient toutes les données relatives à une série c'est-à-dire l'ensemble des valeurs pour chaque catégorie.

La *VerticalBarChartRenderer* contient l'ensemble des séries d'un histogramme. Cette classe est responsable de la représentation d'un histogramme dans ce graphique complexe, position de l'axe et décalage.

La classe *CombinedVerticalBarChartXYY* permet de dessiner le graphique de la taille (hauteur, largeur) souhaitée. Pour cela il faut lui donner les informations sur les axes X et Y, ainsi que l'ensemble des VerticalBarChartRenderer.

```
// Création du Conteneur de série
CategoryValueSeriesContainer categoryValueSeriesContainer = new CategoryValueSeriesContainer();

// 1 - Création de l'ensemble des catégories présentée dans le graphique
Vector allCategories = new Vector();
allCategories.addElement(new Category("id1","Equip 1 TR"));
allCategories.addElement(new Category("id2","Equip 2 TR"));
allCategories.addElement(new Category("id3","Equip 3 TR"));
allCategories.addElement(new Category("id4","Equip 4 TR"));
allCategories.addElement(new Category("id5","Equip 5 TR"));

// 2 - Création du premier histogramme lié à l'axe des Y de gauche
// Création du conteneur de série
CategoryValueSeriesContainer categoryValueSeriesContainer = new CategoryValueSeriesContainer();
// Définition des catégories de l'histogramme
categoryValueSeriesContainer.addCategory(allCategories.elements());
// Série 1 (construction de la série et ajout des valeurs pour chaque catégorie)
CategoryValueSeries categoryValueSeries = new CategoryValueSeries("S1", "");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 18));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 28));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 15));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 20));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 50));
// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);
// Série 2 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S2", "");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 18));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 42));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 15));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 24));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 15));
// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);
// Création du premier histogramme
VerticalBarChartRenderer renderer1 = new VerticalBarChartRenderer(categoryValueSeriesContainer);


// 3 - Création du deuxième histogramme lié à l'axe des Y de droite
// Création du conteneur de série
categoryValueSeriesContainer = new CategoryValueSeriesContainer();
// Définition des catégories de l'histogramme
categoryValueSeriesContainer.addCategory(allCategories.elements());
// Série 1 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S1", "");
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 6));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 6));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 3));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 8));
```

```
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 8));
// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);
// Série 2 (construction de la série et ajout des valeurs pour chaque catégorie)
categoryValueSeries = new CategoryValueSeries("S2", "");//serie 1
categoryValueSeries.addCategoryValue(new CategoryValue("id1", 6));
categoryValueSeries.addCategoryValue(new CategoryValue("id2", 4));
categoryValueSeries.addCategoryValue(new CategoryValue("id3", 3));
categoryValueSeries.addCategoryValue(new CategoryValue("id4", 2));
categoryValueSeries.addCategoryValue(new CategoryValue("id5", 3));
// Ajout de la série dans le conteneur
categoryValueSeriesContainer.addCategoryValueSeries(categoryValueSeries);
// Création du deuxième histogramme
VerticalBarChartRenderer renderer2 = new VerticalBarChartRenderer(categoryValueSeriesContainer);
// Histogramme lié à l'axe des Y de droite

renderer2.setIsBasedOnAxisRight(true);
// Positionne le nombre de pixel de décalage d'affichage par rapport au prelier histogramme
renderer2.setNbPixelToMoveRendererStartPosition(10);

// 4 - Génération du graphique au format JPEG
// Création de l'objet réalisant le graphique
int imageHeight = 500; int imageWidth = 900;
CombinedVerticalBarChartXYY combinedVerticalBarChart = new CombinedVerticalBarChartXYY(
        imageWidth, imageHeight, new HorizontalCategoryAxis(""),
        new VerticalNumberAxis("Flow rate (kg/h)"), new VerticalNumberAxisRight("Conso (l)"));
// Ajout  histogramme 1
combinedVerticalBarChart.addVerticalBarChartRenderer(renderer1);
// Ajout  histogramme 2
combinedVerticalBarChart.addVerticalBarChartRenderer(renderer2); //ajout renderer 2
// Ajout de la ligne horizontale d'information
combinedVerticalBarChart.addHorizontalLineInfo(new HorizontalLineInfo("Custom Flow rate", 50));
// Génération du fichier JPEG
combinedVerticalBarChart.drawJPEG("test.jpg"); //draw as JPEG in file test.jpg
```

8. Pie Chart

8.1 Description

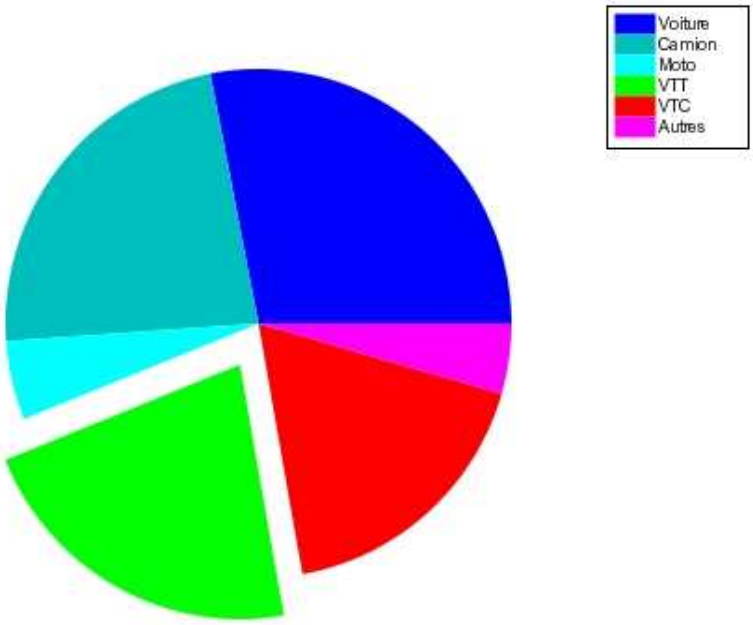
La classe ***ExplodedPieChart*** permet la génération de secteurs plus communément appelés « camembert ». Un camembert est la représentation de plusieurs valeurs de façon proportionnelle.

Les entités ou secteurs représentés sur ce type de graphique sont définies par des triplets, il s'agit d'un identifiant de secteur, d'un intitulé noté dans la légende et d'une valeur numérique positive. Un tel triplet correspond à une série et, les séries d'un même graphique sont regroupées dans un container.

Il est possible de générer des camembert 2D ou 3D, de faire ressortir une ou plusieurs secteurs du camembert, de faire figurer sur le graphique les labels ou les valeurs …

8.2 Exemple

L'exemple (camembert 2D) ci-après montre une répartition relative à l'utilisation de différents modes de transport. Ces modes (Voiture, Camion, Moto ; VTT, VTC, Autres) sont au nombre de six et en conséquence il y a six séries de valeurs.



8.3 Code de l'exemple

La classe *ValueSeriesContainer* contiendra toutes les séries de valeurs.

La classe *ValueSerie* contient la valeur d'une série, mais aussi l'intitulé de cette série dans la légende.

La classe *PieChart* permet de dessiner le graphique de la taille (hauteur, largeur) souhaitée. Pour cela il faut lui donner l'ensemble des séries.

```
// Création du Conteneur de série
ValueSeriesContainer valueSeriesContainer = new ValueSeriesContainer();

// Ajout des séries

valueSeriesContainer.addValueSeries( new PieSeries( "S6", "Voiture",    110,    0  ) );
valueSeriesContainer.addValueSeries( new PieSeries( "S5", "Camion",     90,     0  ) );
valueSeriesContainer.addValueSeries( new PieSeries( "S4", "Moto",       20,     0  ) );
valueSeriesContainer.addValueSeries( new PieSeries( "S3", "VTT",        85,     20 ) ); // mise en valeur de ce secteur
valueSeriesContainer.addValueSeries( new PieSeries( "S2", "VTC",        70,     0  ) );
valueSeriesContainer.addValueSeries( new PieSeries( "S1", "Autres",     15,     0  ) );

// Taille de l'image
int imageHeight = 500; int imageWidth = 530;

ExplodedPieChart pieChart = new ExplodedPieChart(imageWidth, imageHeight, valueSeriesContainer);

// Génération du graphique au format JPEG
pieChart.drawJPEG("test.jpg");

// Génération du graphique au format SVG
pieChart.drawSVG ("test.jpg");
```

8.4 Quelques méthodes de mise en forme

//Passage à un camembert 3D

```
pieChart.set3Dprofondeur( 20 );
```

//Passage à un camembert allongé sur l 'axe horizontal

```
pieChart.setCircular( false );
pieChart.setVerticalEllipse( false );
```

//Légende sur les secteurs du graphique

```
pieChart.setPieWithTags( true );
```

// Type de légende sur les secteurs du graphiques ; valeur par défaut = TAG_VALUE

```
pieChart.setTypeOfTag( ExplodedPieChart.TAG_LABEL );
```

// Valeur minimale des secteurs ayant une légende ; par défaut cette valeur est fixée à 2.

```
pieChart.setMinPercentToDisplayTag( 5 );  // la valeur de la série « Autres » n'apparaîtra pas.
```

// Largeur de la marge entre le graphique et les légendes de secteurs ; par défaut cette valeur est fixée à 1.

```
pieChart.setTagOnPieWidth( 10 );
```

// Positionnement de la légende

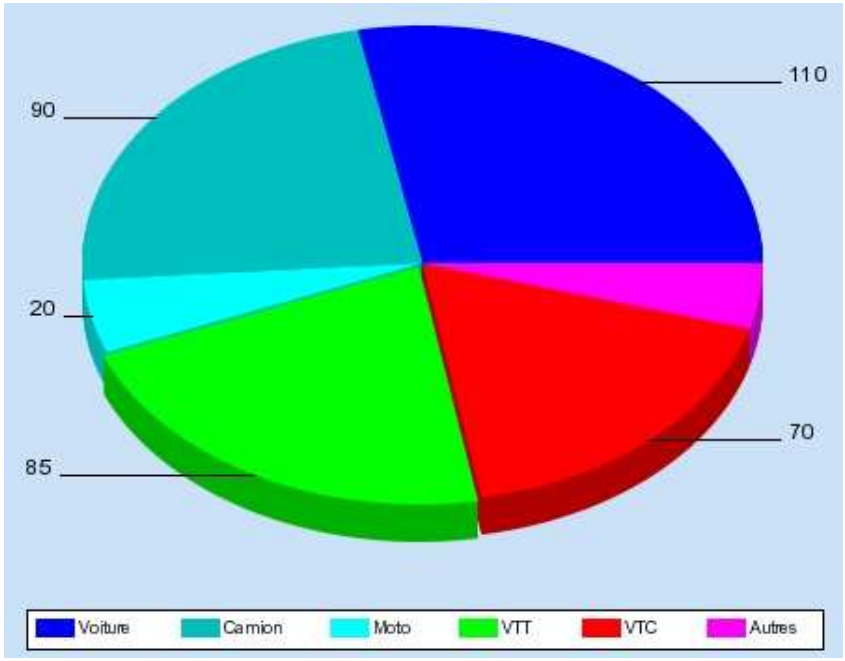
- **LEGEND_DISPLAY_MODE_BOTTOM**
- **LEGEND_DISPLAY_MODE_LEFT (par défaut)**
- **LEGEND_DISPLAY_MODE_TOP**

```
pieChart.setLegendDisplayMode( Legend.LEGEND_DISPLAY_MODE_BOTTOM );
```

// Couleur de fond

```
pieChart.setGraphicBackgroundColor( new Color( 202, 225, 247 ) );
pieChart.setImageBackgroundColor( new Color( 202, 225, 247 ) );
pieChart.setBackgroundColorVisible( true );
```

Le résultat obtenu est l'image suivante :



9. Chart Animation

9.1 Description

OFC-Charts permet d'animer des graphiques pour les présenter dans une page WEB par exemple. L'animation consiste en une succession d'images faisant apparaître progressivement les différents éléments du graphique.

Plus précisément, les graphiques composés d'un ensembles de séries tels que les courbes, les histogrammes , … sont animés en visualisant progressivement les différentes séries.

La classe ***ChartAnimator*** permet la génération de graphiques animés, le résultat de l'animation est un fichier SVG. Cette classe prend en compte des paramètres d'animation tels que la durée de l'animation ainsi que le nombre de répétition de cette animation.

9.2 Code exemple

Ci-dessous, la séquence de code permettant la génération d'un fichier SVG contenant l'animation de l'histogramme empilé défini au paragraphe 6.2. Dans cet exemple, l'animation dure 3 secondes et elle est répétée indéfiniment.

```
// Création du l'objet permettant d'animer un graphique
ChartAnimator animator = new ChartAnimator(verticalStackedBarChart);

// Durée de l'animation, elle est exprimée en minutes et en seconde (ici 3 secondes)
 int nbMinute = 0;
int nbSecond = 3;

// Nombre de répétition de l'animation (ici on recommence l'animation à l'infini)
int nbRepeat =   SVGAnimationContext.INDEFINITE_REPEAT;

// Création du fichier SVG animé
animator.drawSVG(new FileOutputStream("testAnimateVerticalStackedBarChart.svg"),  nbMinute,  nbSecond, nbRepeat);
```

De la même manière, on pourrait animer un « VerticalBarChart », « VerticalLineChart », « VerticalLine Plot », « PieChart ».

10. Contour 2D

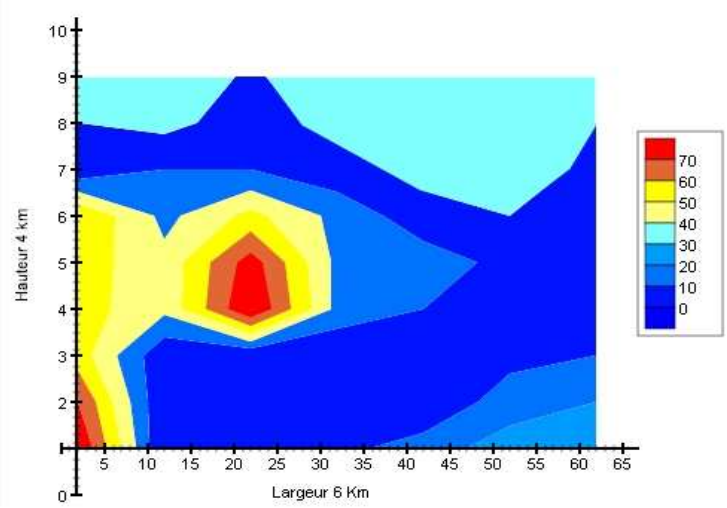
10.1 Description

La classe ***Contour2D*** permet à partir d'un maillage carré, régulier ou non, de générer les courbes d'iso valeur.

Ainsi, par exemple, pour un maillage représentant un modèle numérique de terrain, c'est-à-dire un maillage pour lequel chaque nœud correspond à une altitude, l'outil génère les courbes de niveau ou « isoligne ».

10.2 Exemple

La réalisation de cet exemple nécessite l'utilisation de la classe *Contour2D* avec en paramètre le maillage correspondant (ici fichier « contour2dTest.txt ») ainsi que le pas de génération des courbes d'iso valeur. La classe *Contour2D* génère alors le résultat suivant dans un fichier JPEG ou SVG.



10.3 Code exemple

```
// Dimensions de l'image
int imageHeight = 400;  int imageWidth = 500;

// Description de l'axe horizontal
HorizontalNumberAxis axisX = new HorizontalNumberAxis("Largeur 6 Km");
axisX.setLabelPosition(HorizontalNumberAxis.LABEL_POSITION_HORIZONTAL_RIGHT_CENTER); axisX.disableArrow();

// Description de l'axe vertical
VerticalNumberAxis axisY = new VerticalNumberAxis("Hauteur 4 km");
axisY.setLabelPosition(VerticalNumberAxis.LABEL_POSITION_VERTICAL_LEFT_CENTER);
axisY.disableArrow();

// Positionnement des valeurs de début, fin et le pas
double _step = 10;  double _start = 70;  double _end = 0;
```

```
try {
        // Création de l'objet 2D Contour : les données sont récupérées à partir d'un fichier ( contour2dTest.txt )
        Contour2D c2d = new Contour2D(imageWidth, imageHeight, axisX, axisY,
                                        HaGrid.getGrid("contour2dTest.txt"),  _start, _end,  _step);
        // Création de l'image
        c2d.drawJPEG("contour2D.jpg");
} catch (Exception ex) {      ex.printStackTrace();    }
```

10.4 Données

Les données peuvent être fournies à l'aide d'un fichier ou directement dans l'application.

```
        IrregularGrid grid = new IrregularGrid(); // ou RegularGrid grid = new RegularGrid();
        Vector allLine = new Vector();
        Vector allGridNodeOfLine = new Vector();
// new GridNode(x, y, z) ; pour y = 0.5
// Ajout d'un nœud de la ligne
        allGridNodeOfLine.addElement(new GridNode(10, 0.5, -73.3));
        allGridNodeOfLine.addElement(new GridNode(30, 0.5, -74.8));
        allGridNodeOfLine.addElement(new GridNode(50, 0.5, -75.1));
        allGridNodeOfLine.addElement(new GridNode(70, 0.5, -74.8));
        allGridNodeOfLine.addElement(new GridNode(90, 0.5, -73.3));
// Construction d'une ligne
        allLine.addElement(allGridNodeOfLine);
// new GridNode(x, y, z) ; pour y = 1.5
        allGridNodeOfLine = new Vector();
        allGridNodeOfLine.addElement(new GridNode(10, 1.5, -73.4));
        allGridNodeOfLine.addElement(new GridNode(30, 1.5, -74.9));
        allGridNodeOfLine.addElement(new GridNode(50, 1.5, -75.2));
        allGridNodeOfLine.addElement(new GridNode(70, 1.5, -74.9));
        allGridNodeOfLine.addElement(new GridNode(90, 1.5, -73.4));
        allLine.addElement(allGridNodeOfLine);
// new GridNode(x, y, z) ; pour y = 2.5
        allGridNodeOfLine = new Vector();
        allGridNodeOfLine.addElement(new GridNode(10, 2.5, -73.5));
        …
// Construction de la grille complète des données
        for(int i=allLine.size() - 1; i >= 0; i--){
                grid.addLine((Vector)allLine.elementAt(i));
        }
```

Dans l'exemple le fichier a la structure suivante :

Première ligne = valeurs des X

Deuxième ligne = valeurs des Y

//commentaire

Bloc des valeurs Z (température, altitudes …)

//commentaire

Bloc des valeurs Z (température, altitudes …)

La classe `HaGrid` lit ce fichier et affecte les valeurs x, y et z :

```
            IrregularGrid grid = new IrregularGrid();
            Vector allLine = new Vector();
            for(int i= 0; i<nbLine; i++){
              Vector allGridNodeOfLine = new Vector();
              Vector lineData = (Vector)_temp.elementAt(i);
              double y = ((Double)_y.elementAt(i)).doubleValue();;
              for(int j=0; j< nbColumn; j++){
                double x = ((Double)_x.elementAt(j)).doubleValue();;
                double z = ((Double)lineData.elementAt(j)).doubleValue();
// Ajout d'un nœud de la ligne
                allGridNodeOfLine.addElement(new GridNode(x, y, z));
              }
// Construction d'une ligne
              allLine.addElement(allGridNodeOfLine);
            }
// Construction de la grille ligne par ligne
          for(int i=allLine.size() - 1; i >= 0; i--){
                  grid.addLine((Vector)allLine.elementAt(i));
          }
// Construction de la grille complète des données
          allGridNodeOfLine.addElement(new GridNode(x, y, z));
```

10.5 Animation

L'animation de plusieurs *Contour2D* peut s'avérer intéressante pour visualiser l'évolution d'un phénomène dans le temps.

La classe *Contour2DAnimator* rend cette tâche très facile. Il suffit de lui donner tous les Contour2D, sous une **forme SVG**, dans l'ordre chronologique de l'animation et la classe *Contour2DAnimator* fournira un fichier **SVG animé**.

11. Animation

11.1 Différentes possibilités

OFC-Charts permet l'animation des graphiques. Les graphiques animés sont tous générés en *SVG*.

Plusieurs possibilités sont offertes :

- Les animations avec des séquences d'images embarquées.
- Les animations temps réel.

11.2 Animation embarquée

Les animations avec des séquences d'images embarquées, sont des animations dont la description est entièrement contenue dans le fichier *SVG.* Plusieurs mécanismes d'animation sont possibles :

- L'animation des séries du graphique qui seront rendues visibles de manière progressive (voir la classe ***ChartAnimator***)
- L'animation de plusieurs *Contours2D* enchaînés les uns à la suite des autres (voir la classe ***Contour2DAnimator***)
- L'animation de plusieurs *Charts* enchaînés les uns à la suite des autres (voir la classe ***MultiCharAnimator***)

Pour réaliser des animations embarquées, Il faut :

- donner (via la méthode *add*) à une des trois classes ci-dessus le graphique ou les graphiques devant être animés
- appeler la méthode *drawSVG* permettant la génération du *svg* animé avec les informations d'animation souhaitées (durée d'animation, nombre de répétition de l'animation)

Exemple :

```
int minute  =  0;
int second =  3;
int nbOfrepeat =  SVGAnimationContext.INDEFINITE_REPEAT;

// ChartAnimator
ChartAnimator animator = newChartAnimator(combinedVerticalBarChartXY);
animator.drawSVG ( new FileOutputStream("testCombinedVerticalBarChartXY.svg"),
                        minute,
                        second,
                        nbOfrepeat);
```

```
// Contour2DAnimator
 Contour2DAnimator animator = new Contour2DAnimator();
 double zStep = 5;
 double zSart = -90;
 double zEnd = 0;

 for (int i=0 ; i < 9 ; i++){
         animator.add( new Contour2D (imageWidth, imageHeight, axisX, axisY,
                          HaGrid.getGrid ("courbe" + i + ".txt"), zStart, zEnd, zStep));
  }
 animator.drawSVG ( new FileOutputStream ("Contour2DAnimate.svg"),
                        minute,
                        second,
                        SVGAnimationContext.INDEFINITE_REPEAT);
```

11.3 Animation temps réel

L'animation d'une partie ou de la totalité du graphique peut être réalisée de façon externe.

Pour cela *OFC-Charts* propose un moyen de spécifier une partie du graphique comme étant dynamique et pouvant être mise à jour de façon externe via une *URL* donnée. Le graphique est alors composé d'une partie statique et d'une partie dynamique. La mise a jour de la partie dynamique du graphique sera réalisée de façon régulière au cours du temps.

La fluidité de l'animation et le rafraîchissement de la partie dynamique du graphique sont entièrement pris en charge par *OFC-Charts* qui permet ainsi la gestion de graphique dont les données évoluent en temps réel.

Pour réaliser des animations temps réel, il suffit, par exemple, de réaliser une *servlet* avec deux actions :

- La première action de la *servlet* fournit le *SVG* de la partie statique du graphique et le contexte de l'animation de la partie dynamique (c'est-à-dire le temps de rafraîchissement et l'*URL* pour localiser la partie dynamique). Le code de la partie statique du graphique peut correspondre à chacun des exemples précédemment cités, par exemple le code de réalisation d'un *VerticalBarChart* pour lequel seule la partie que l'on souhaite rendre statique est visible (par exemple le fond quadrillé) et ceci via les possibilités *d'OFC-Charts* de rendre invisible des parties du graphique.

- La deuxième action de la *servlet* fournit uniquement la partie dynamique courante du graphique. Le code de la partie dynamique du graphique peut correspondre à chacun des exemples précédemment cités, par exemple le code de réalisation d'un *VerticalBarChart* pour lequel seules les barres d'histogramme sont visibles et ceci via les possibilités *d'OFC-Charts* de rendre invisible des parties du graphique.