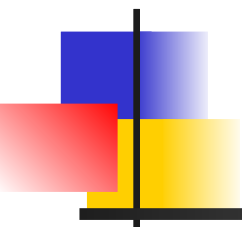




# Éléments avancés

---

Groupes  
Sous-rapports  
Datasets, Listes et Tableaux  
Graphiques  
Tableaux croisés  
Annexes



# Les groupes



# Groupe

- Les groupes permettent de regrouper les données d'un rapport selon des **critères de rupture** spécifiés via des expressions.
- La définition d'un groupe crée généralement de nouvelles bandes :
  - une **entête** (`<groupFooter>`)
  - **un pied de groupe** (`<groupHeader>`)  
*Rapport → Clic droit → Créer groupe*
- Lors de la fusion de données, *JasperReport* teste toutes les expressions de groupe définies afin de déterminer si une rupture de groupe survient.  
Si c'est le cas, il insère l'entête et le pied de groupe associés.



# Example

```
<group name="CITY">
  <groupExpression><![CDATA[$F{CITY}]]></groupExpression>
  <groupHeader>
    <band height="37">
      <frame>
        <reportElement mode="Opaque" x="0" y="7" width="555" height="24"
forecolor="#B89F7D" backcolor="#000000"/>
        <textField isStretchWithOverflow="true">
          <reportElement style="SubTitle"
isPrintRepeatedValues="false" x="2" y="0" width="479" height="24"
forecolor="#FFFFFF"/>
          <textElement><font isBold="false"/></textElement>
          <textFieldExpression class="java.lang.String">
            <![CDATA[$F{CITY}]]>
          </textFieldExpression>
        </textField>
      </frame>
    </band>
  </groupHeader>
  <groupFooter>
    <band height="6"/>
  </groupFooter>
</group>
```



# Ordre

---

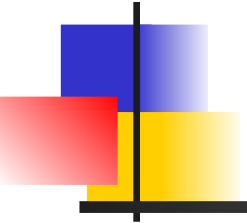
- Le moteur n'effectue pas de tri sur les enregistrements.
- => Il faut donc toujours ajouter une clause **ORDER** dans la requête SQL
- Dans le cas d'une source XML, il faut alors demander explicitement à JasperReport d'effectuer le tri.



# Imbrication des groupes

---

- Le nombre de groupes définis dans un rapport est illimité.
- L'ordre est important car les groupes sont contenus les uns dans les autres.
- Lorsqu'un groupe rencontre une rupture, tous les groupes contenus sont réinitialisés.



# Attributs d'un groupe

---

- ***groupExpression*** : L'expression
- ***isStartNewColumn*** : Si true, un saut de colonne est effectué à chaque rupture. Dans le cas d'un rapport avec une seule colonne, c'est un saut de page qui est effectué
- ***isStartNewPage*** : Si true, un saut de page est effectué à chaque rupture
- ***isResetPageNumber*** : Saut de page + réinitialisation du numéro de page
- ***isReprintHeaderOnEachPage*** : L'entête de groupe est rappelé à chaque changement de page
- ***minHeightToStartNewPage*** : Hauteur minimale déclenchant le saut de page

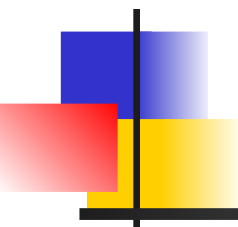


# Variables de groupe

---

- Lors de l'ajout d'un groupe, *JasperReport* crée automatiquement une variable :  
**<groupName>\_COUNT**
- C'est une variable d'itération
- Utilisée en bas de groupe, elle donne le nombre d'enregistrements pour le groupe





# Sous-rapports



# Introduction

---

- Les **sous-rapports** permettent la création de rapports complexes et la réutilisation.
- Ils sont souvent utilisés dans des rapport de type « master-detail » ou lorsque la structure d'un rapport n'est pas suffisante pour décrire la complexité du document à générer.
- Un sous-rapport est un rapport normal inclus dans un autre rapport.  
=> Tout rapport peut être utilisé comme sous-rapport dans un autre rapport sans modification.
- Il n'y a pas de limite d'imbrication des sous-rapports.
- Les listes/tableaux proposent une alternative simplifiée des sous-rapports



# Utilisation

---

- Un sous-rapport est désigné complètement indépendamment d'un autre rapport.
- C'est au moment de son intégration dans le rapport parent (balise **<subReport>**) que les informations suivantes doivent être fournies :
  - **Référence** au fichier *.jrxml* du sous-rapport
  - La **connexion** à utiliser
  - Les **paramètres** d'entrées à fournir
  - Les **valeurs de retour** que l'on veut utiliser dans le rapport parent



# Marges et dimensions

---

- Les marges d'un sous-rapport sont généralement positionnées à 0.
- Lors de son utilisation, il est possible d'indiquer une taille à l'élément *<subReport>*
  - Ce n'est pas cette taille qui est utilisée lors de la génération.  
La taille est définie dans le sous-rapport.
  - L'élément *<subReport>* ne permet donc que de positionner le sous-rapport dans le rapport parent
    - Il est cependant conseillé de positionner la largeur de l'élément à la même valeur que la taille du sous-rapport pour « préserver » l'aspect WYSWIG.



# Référence

- La référence au fichier se fait via le sous-élément **<subReportExpression>** et son attribut **class** pouvant prendre les valeurs suivantes :
  - **java.lang.String** : L'emplacement du fichier Jasper. Par rapport au système de fichier ou au CLASSPATH ou une URL. L'attribut **isUsingCache** permet de ne pas recharger à chaque fois le sous-rapport.
  - **java.io.File** : Expression Java retournant un objet File
  - **java.net.URL** : Une URL
  - ...



# Source de données

---

- En général, il y a 3 possibilités concernant la source de données :
  - Le sous-rapport utilise une requête SQL et nécessite la **même connexion JDBC** utilisée par le parent
  - Le sous-rapport utilise un **autre type de source de données** comme un fichier XML par exemple
  - Le sous-rapport n'utilise **pas de source de données** et est utilisé pour simuler l'inclusion de portions statiques de document (entête, pied de page, ...)



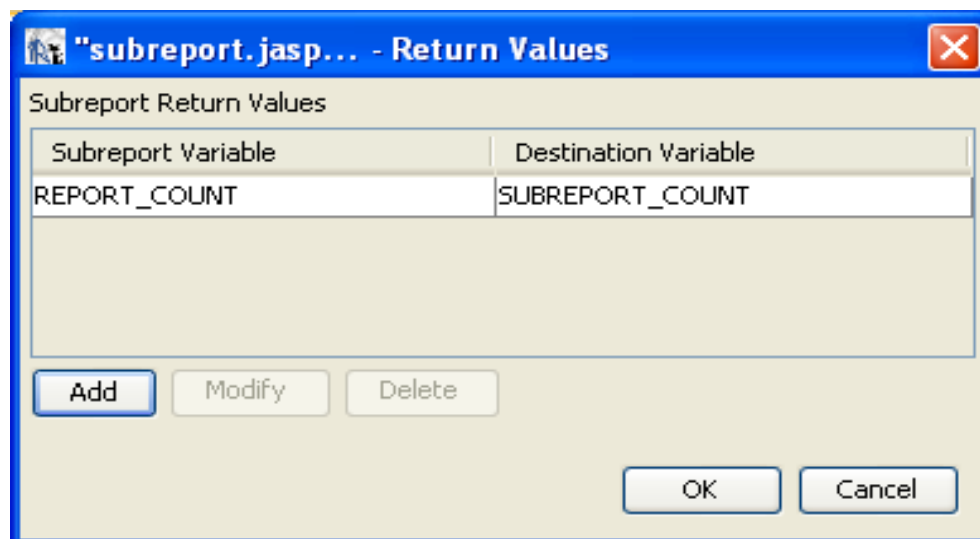
# Paramètres d'un sous-rapport

---

- Les paramètres d'un sous-rapport peuvent être fournis :
  - Soit sous forme de *Map* par l'élément **<parametersMapExpression>**.  
On peut ainsi passer tous les paramètres du parent en utilisant  
`$P{REPORT_PARAMETERS_MAP}`
  - Soit en utilisant l'élément **<subreportParameter>** pour chaque paramètre. (ces éléments surchargent la map éventuellement spécifiée).

# Valeurs de retour

- Il est possible de mettre à jour des variables du rapport parent avec les variables calculées du sous-rapport
  - Il suffit alors de définir les correspondances entre la variable de sous-rapport et la variable du parent.
  - Des fonctions d'agrégation peuvent également être appliquées



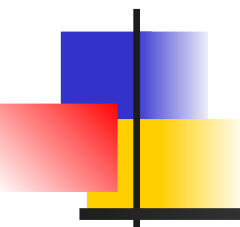




# Méthode à suivre

---

- Définir une variable dans le rapport parent dont la méthode de calcul est « **System** »
- Définir la correspondance entre la variable créée et une des variables du sous-rapport. Les types devant correspondre
- Si la valeur de la variable est affichée dans la même bande que le sous-rapport, alors l'attribut « **Evaluation time** » du champ texte doit être positionné à « **band** »



# Datasets, Listes et Tables



# Datasets / Jeux de données

---

- Les graphiques et tableaux croisés ont souvent besoin d'utiliser des données non fournies par la requête principale du rapport
- Grâce à la notion de **datasets**, différents graphiques ou tableaux croisés peuvent être inclus sans utiliser de sous-rapports
- Un dataset est un mélange entre :
  - Un jeu de données : on va pouvoir définir de nouvelles requêtes
  - et un sous-rapport : il contient des paramètres des champs, des variables et des groupes mais pas d'information de disposition.

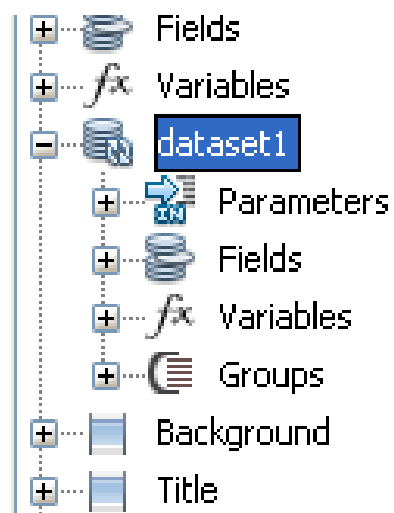


# Élément *<subdataset>*

---

- Les datasets sont déclarés à l'intérieur de la balise *<jasperReport>* via l'élément ***<subdataset>***.
- Le nombre de datasets n'est pas limité.
- Les propriétés principales du *dataset* sont :
  - La requête permettant d'avoir un autre jeu de données
  - Les champs, paramètres, variables et groupes qui seront utilisables

# Outline





# Dataset run

---

- Un *dataset* peut être associé à un ou plusieurs éléments itératifs : liste, tableau, graphique ou tableau croisé par l'intermédiaire d'un **dataset run**
- La configuration d'un *dataset run* est similaire à la configuration d'un sous-rapport, il faut spécifier :
  - La connexion à la source de données
  - Le mapping des paramètres
  - Les éventuels valeurs de retour
- Tous les champs, paramètres et variables du dataset peuvent alors être utilisés dans l'élément itératif

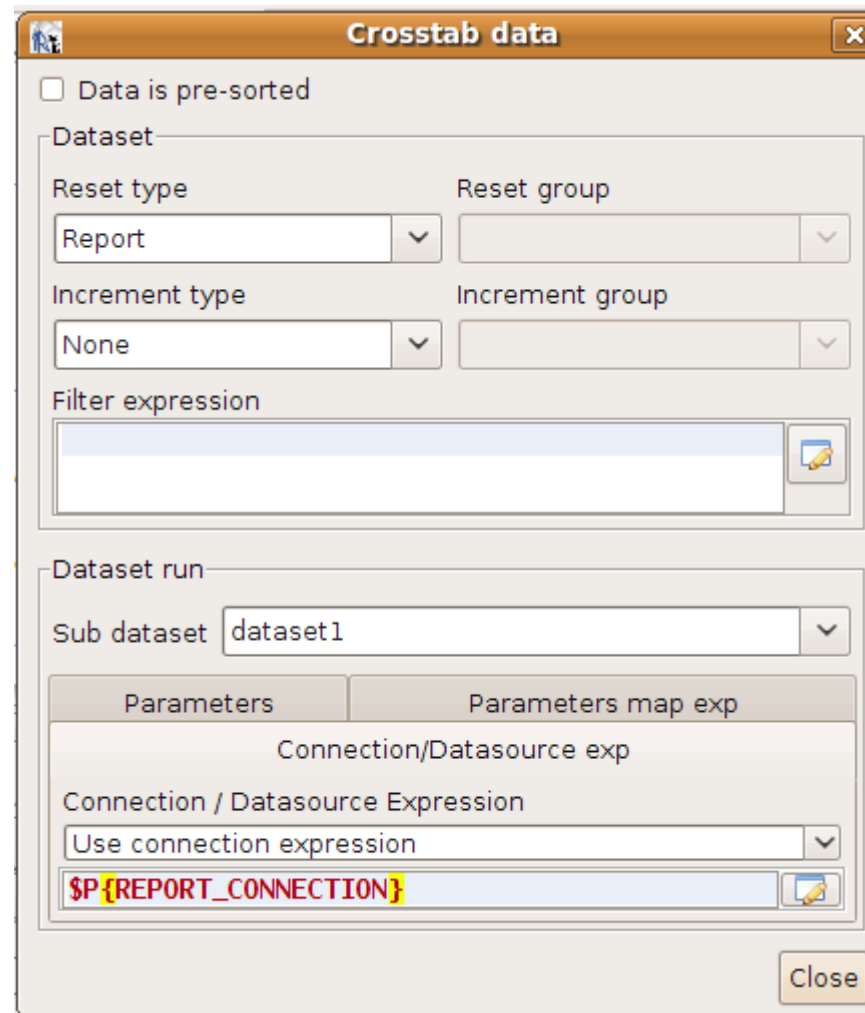


# Contexte du dataset

---

- Les variables, paramètres et groupes du *dataset* ne peuvent être utilisés que par les éléments itératifs qui utilisent le *dataset*. Ils ne sont pas visibles dans le rapport principal
- Les groupes ne sont donc utilisés que pour le calcul des variables du dataset. Ils ne provoquent pas d'espaces d'impression supplémentaires

# Dataset run



The screenshot shows a dialog box titled "Crosstab data" with a close button (X) in the top right corner. The dialog is divided into several sections:

- Data is pre-sorted:** A checkbox that is currently unchecked.
- Dataset:** A section containing:
  - Reset type:** A dropdown menu with "Report" selected.
  - Reset group:** An empty dropdown menu.
  - Increment type:** A dropdown menu with "None" selected.
  - Increment group:** An empty dropdown menu.
  - Filter expression:** A text area that is empty, with a small icon (a notepad and pencil) to its right.
- Dataset run:** A section containing:
  - Sub dataset:** A dropdown menu with "dataset1" selected.
  - Parameters:** A tab that is currently selected, showing a text area with the expression `$P{REPORT_CONNECTION}` and a small icon (a notepad and pencil) to its right.
  - Parameters map exp:** A tab that is currently unselected.
  - Connection/Datasource exp:** A section containing:
    - Connection / Datasource Expression:** A dropdown menu with "Use connection expression" selected.
    - Text area:** A text area containing the expression `$P{REPORT_CONNECTION}` and a small icon (a notepad and pencil) to its right.

A "Close" button is located at the bottom right of the dialog.





# Autres propriétés

---

- En dehors de l'attribut ***name*** qui doit être unique dans le rapport, un dataset peut définir :
  - ***scriptletClass*** : Une classe scriptlet spécifique
  - ***resourceBundle*** : Des fichiers de localisation des labels
  - ***whenResourceMissingType*** : Le comportement à adopter si une clé du bundle manque
  - Des expressions de **filtre** et de **tri**



# Liste

L'élément **Liste** est une version ultra-simplifiée d'un sous-rapport.

Il ne propose pas de variables, ni de bandes

Mais il est associé à une source de données (*dataset*), et a des paramètres éventuels

L'usage d'une liste consiste à « itérer » sur un autre jeu de données que les données principales

L'élément peut se placer dans n'importe quelle section du rapport, l'affichage consiste à imprimer des items sous forme de liste ; les seules latitudes sont :

La taille des items

La direction de remplissage de la liste (Horizontal ou Vertical)



# Table

---

*jasperStudio* propose également l'élément **table** dédié à l'affichage de données tabulaires

L'utilisation d'une table est plus facile que l'utilisation de champs textes individuels avec des bordures.

- Une DTD XML distincte est associée à l'élément table

<http://jasperreports.sourceforge.net/sample.reference/table/index.html>



# Structure d'une table

---

A l'intérieur d'une table, des sections sont déclarées afin de regrouper le contenu :

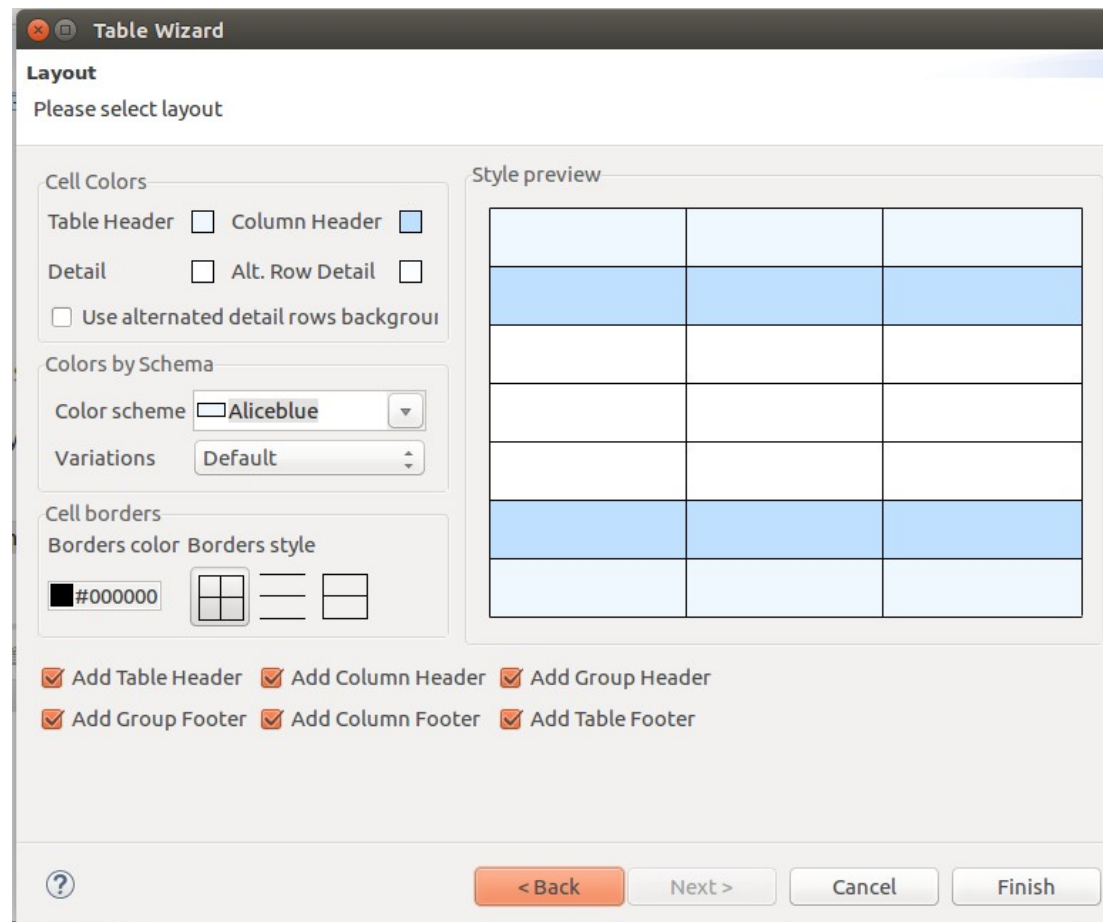
- en-tête et bas de table
- entête et bas de colonne
- entête et bas de ligne
- ainsi qu'un nombre illimité d'en-tête et de bas de groupes imbriqués.

Chaque section est composée d'une liste de **colonnes**.

- Ces colonnes peuvent être groupées
- Elles peuvent être cachées en fonction d'une condition booléenne

A chaque colonne est associée une cellule qui contient un ou plusieurs éléments de contenus.

# Assistant Table



The image shows a 'Table Wizard' dialog box with a title bar containing a close button, a maximize button, and the text 'Table Wizard'. The main area is titled 'Layout' with the instruction 'Please select layout'. It is divided into several sections: 'Cell Colors' with checkboxes for 'Table Header', 'Column Header', 'Detail', 'Alt. Row Detail', and 'Use alternated detail rows background'; 'Colors by Schema' with a 'Color scheme' dropdown set to 'Aliceblue' and a 'Variations' dropdown set to 'Default'; 'Cell borders' with 'Borders color' set to '#000000' and 'Borders style' shown as a grid icon. To the right is a 'Style preview' showing a 6x3 grid with alternating light blue and white rows. At the bottom, there are six checked checkboxes: 'Add Table Header', 'Add Column Header', 'Add Group Header', 'Add Group Footer', 'Add Column Footer', and 'Add Table Footer'. The footer contains a help icon, a '< Back' button, a 'Next >' button, a 'Cancel' button, and a 'Finish' button.

**Table Wizard**


**Layout**  
Please select layout

**Cell Colors**

Table Header ☐ Column Header ☒  
Detail ☐ Alt. Row Detail ☐  
☐ Use alternated detail rows background


**Colors by Schema**

Color scheme   
Variations

**Cell borders**  
Borders color  Borders style 

**Style preview**


☒ Add Table Header ☒ Add Column Header ☒ Add Group Header  
☒ Add Group Footer ☒ Add Column Footer ☒ Add Table Footer

 < Back Next > Cancel Finish



# Cellules

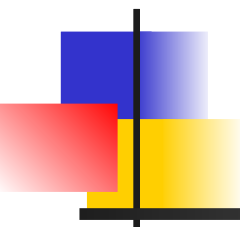
Finalelement, le résultat d'un tableau est une série de cellules se comportant comme des cadres et qui correspondent

- Aux différents en-têtes et bas
- Aux données

Il est possible de fusionner, supprimer, ajouter des cellules.

Il est possible de modifier leur Layout (Vertical par défaut)

Les évaluations différées sont possibles



# Graphiques

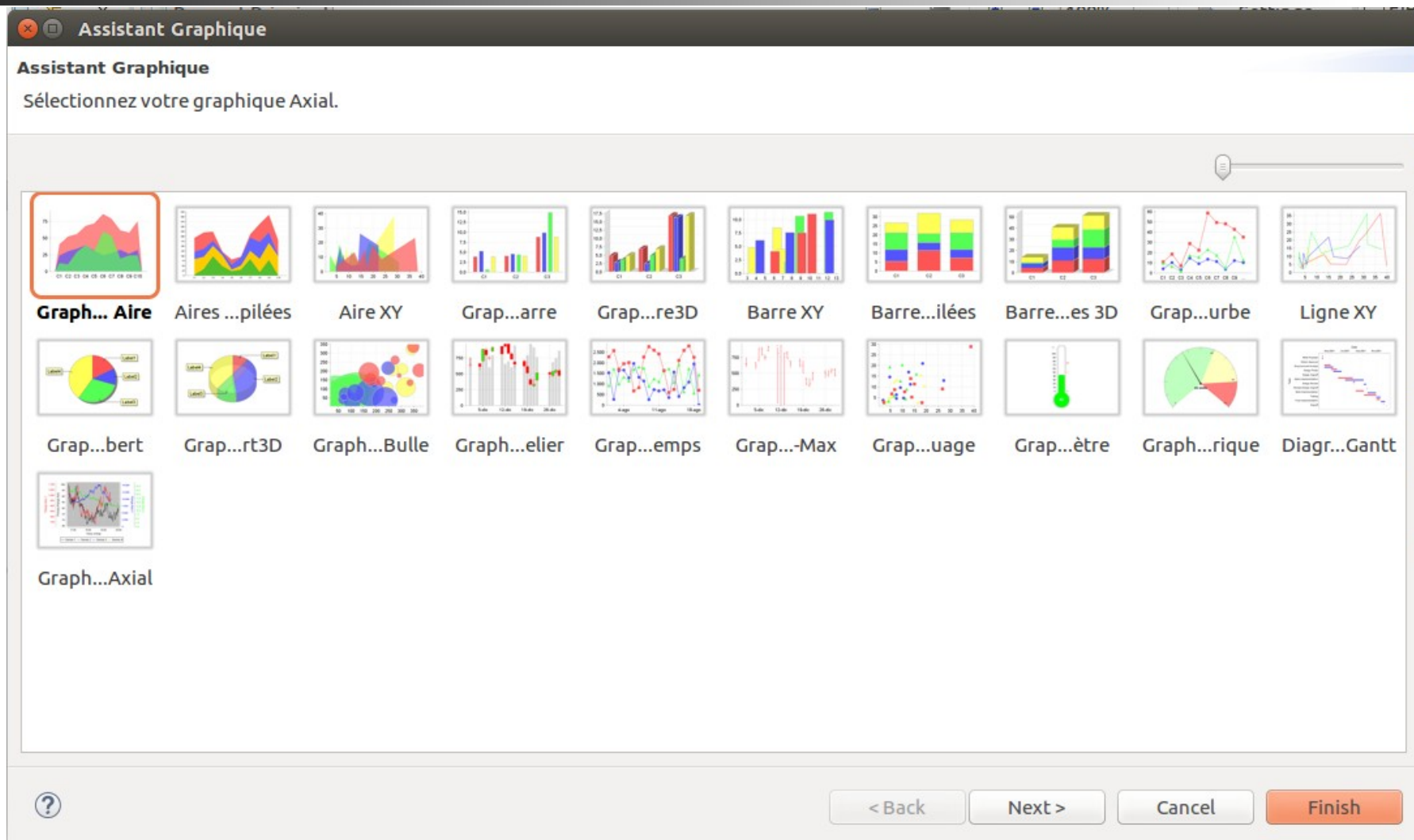


# Graphiques

- JasperReports propose un support pour la création de graphiques s'appuyant sur la librairie *JFreeChart*
- La documentation la plus complète est disponible dans l'ultimate guide de JasperReport et le site *JFreeChart*
- Lors de l'ajout d'un élément **chart**, 4 types d'informations doivent être configurées :
  - Le **type** de graphique
  - Le **jeu de données** utilisé (dataset principal ou un subdataset)
  - Les **expressions** des valeurs du graphique (dépendant du type de graphique)
  - Les paramètres de **rendus** (couleurs, disposition, ...) dépendant également du type de graphique



# Types de graphiques





# Jeux de données

---

- Au moment de la définition du dataset, les attributs communs à tous les types de dataset peuvent être définis :
  - **ResetType** et **ResetGroup** permettent de périodiquement réinitialiser le dataset. Utile si le dataset prend un paramètre (paramètre d'un groupe par exemple)
  - Par défaut, tous les enregistrements sont affichés dans le graphique et tableau croisé. On peut changer ce comportement via les attributs suivants :
    - **IncrementType** et **IncrementGroup** spécifient les moments où des nouvelles valeurs doivent être ajoutées au dataset.
    - L'expression **IncrementWhen** retourne un booléen qui détermine si l'enregistrement courant doit être ajouté au data set. (par défaut tous)



# Propriétés communes

---

- De nombreuses propriétés sont communes à tous les types de graphiques :
  - ***evaluationTime*** et ***evaluationGroup*** :  
Le moment où les expressions du graphique sont évaluées
  - Le titre et sous-titre (expression, police, couleur, position)
  - La légende (police, couleur, position)
  - Le type de rendu (Vectoriel, image)
  - L'orientation des labels
  - ...



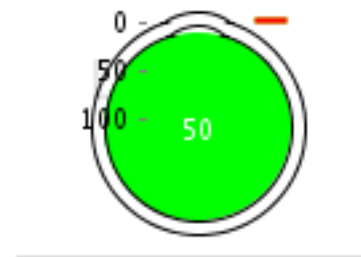
# Données graphiques

---

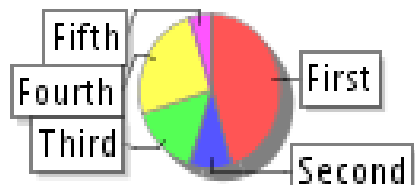
- Les données d'un graphique sont très dépendants du type de graphique
  - Certains peuvent demander de spécifier les valeurs en abscisse, en ordonnée, d'autres des expressions pour évaluer la taille d'une part de camembert, ....
- Mais, en général, il y a 2 types de valeurs :
  - 1 permettant de grouper les données
  - 1 valeur numérique
- De plus, les graphiques permettent de définir plusieurs séries de valeurs numériques donnant lieu à plusieurs tracés sur le même graphique

# Graphique Meter

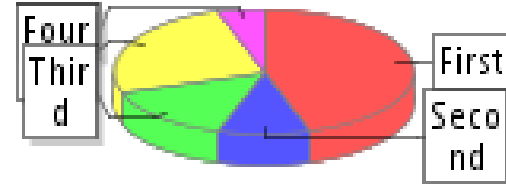
- Les graphiques de type « *Meter* » n'affichent qu'une seule valeur (attribut ***valueExpression***)
- De plus des intervalles permettent de spécifier si cette valeur est jugée bonne, moyenne ou mauvaise (Paramètres de rendu)



# Camemberts



● First ● Second ● Third ● Fourth  
● Fifth



● First ● Second ● Third ● Fourth  
● Fifth

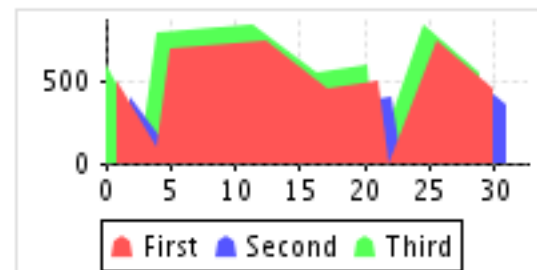
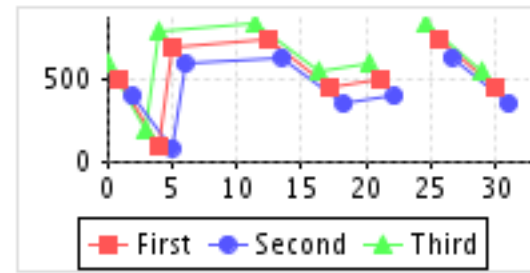
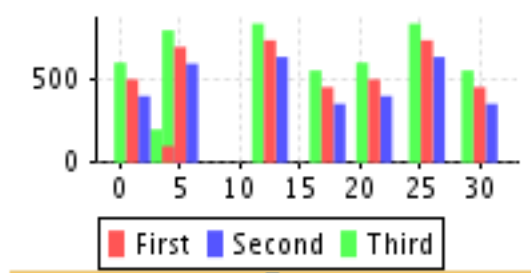


# Données d'un camembert

---

- Les *pieDataSet* sont utilisés par les camemberts. Il faut indiquer :
  - ***serieExpression*** :
  - ***keyExpression*** : Une expression permettant d'identifier de façon unique une part de camembert
  - ***valueExpression*** : L'expression de la valeur
  - ***labelExpression*** : Le label d'une part
- Il est également possible de définir des liens Hypertextes sur chaque portion, de limiter le nombre de part ou d'indiquer un seuil minimal d'affichage

# Graphs X/Y







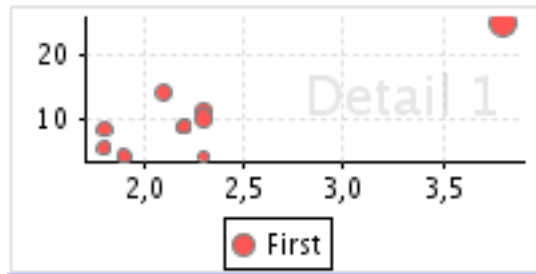
# Graphiques XY

---

- Il est possible de dessiner plusieurs séries sur le même graphe (couleurs différentes)
- Les attributs sont alors :
  - ***seriesExpression*** : L'expression permettant de distinguer les séries. Si une seule série, indiquer une constante
  - ***xValueExpression*** : L'expression en X
  - ***yValueExpression*** : L'expression en Y

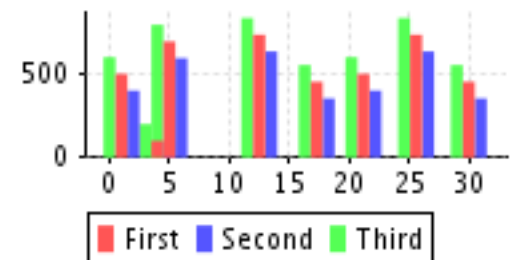
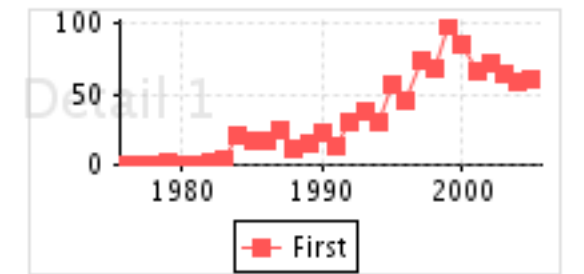
# Graphes XYZ

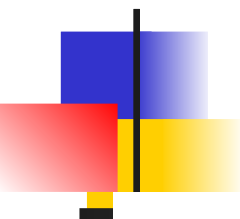
- XYZ ajoute l'attribut ***zValueExpression***



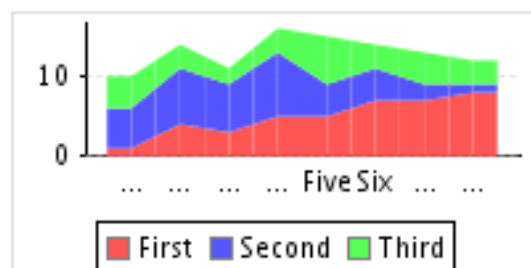
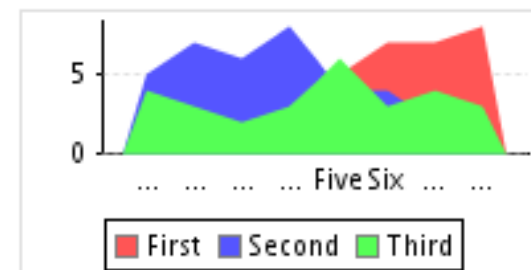
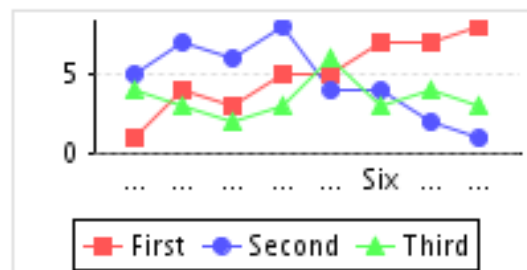
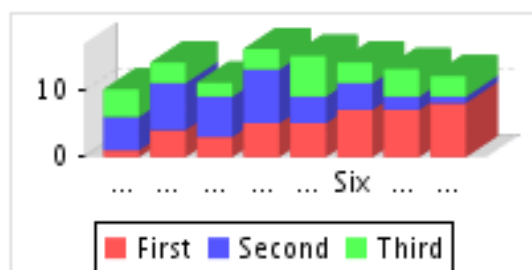
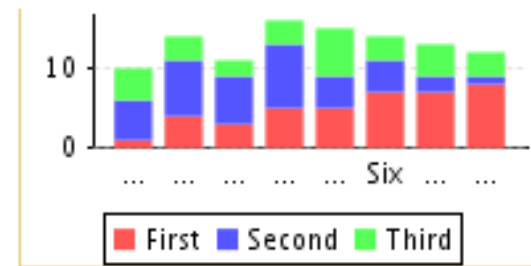
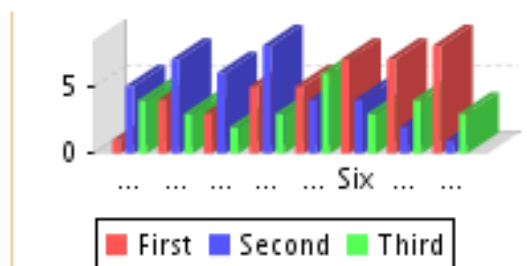
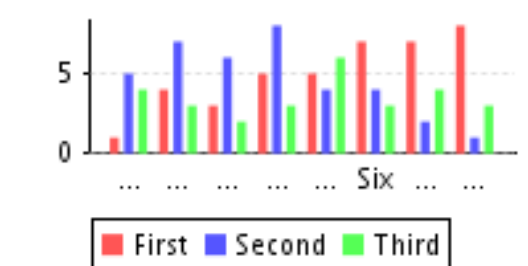
# TimeSeries

- L'axe des abscisses est adapté à des dates





# Category



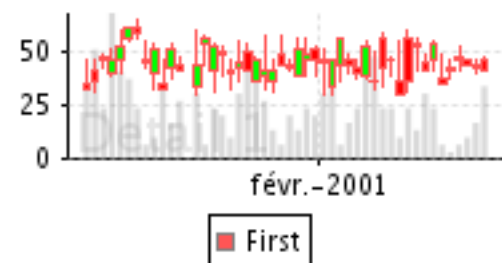
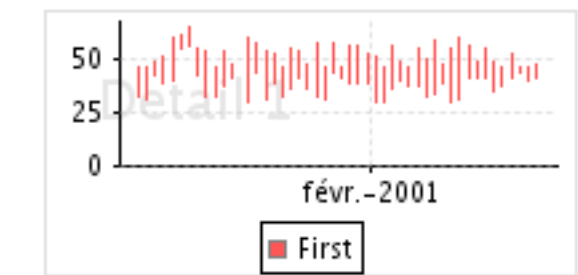


# CategoryDataSet

---

- Les *categoryDataSet* sont adaptés pour comparer différents groupes. L'axe des abscisse correspond à une valeur particulière d'un groupe.
  - ***seriesExpression*** : Permet d'identifier une série (couleur)
  - ***categoryExpression*** : Les différentes valeurs de groupe
  - ***valueExpression*** : La valeur. (Axe des ordonnées)

# Graphiques de type High/Low





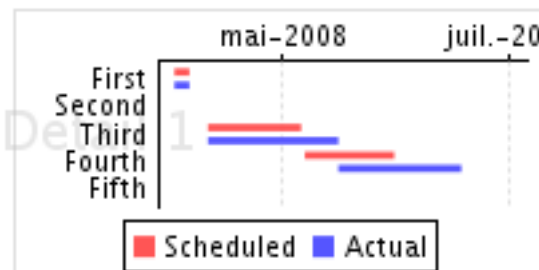
# highLowDataSet

---

- Ces dataset orienté finance permet pour une même abscisse (une date), d'afficher d'une valeur haute et une valeur basse, une valeur d'ouverture et une valeur de clôture.
- Les attributs sont :
  - ***seriesExpression*** : Les couleurs
  - ***dateExpression*** : Axe des abscisse
  - ***highExpression*** : Valeur haute
  - ***lowExpression*** : Valeur basse
  - ***openExpression*** : Valeur d'ouverture
  - ***closeExpression*** : Valeur de clôture
  - ***volumeExpression*** : Volume

# GanttDataSet

- Ces dataset sont adaptés pour les diagrammes de Gant.
- Les attributs sont :
  - ***TaskExpression*** et ***subTaskExpression*** : Pour les tâches
  - ***startDateExpression*** et ***endDateExpression*** : Pour la planification
  - ***percentExpression*** : Pour le pourcentage de réalisation d'une tâche

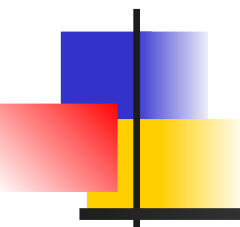






# Attributs de dessin spécifiques

- Des attributs spécifiques de présentation peuvent être indiqués en fonction du type de graphique :
  - Thermomètre : Couleurs du mercure
  - Pie 3D : Facteur 3D
  - Bar, XY Bar, Stacked Bar : Affichage des légendes
  - Bar 3D, Stacked Bar 3D : Affichage des légendes + effet 3D (x offset et y offset)
  - Line, XY Line, Scatter Plot, Time series : Affichage des lignes et des formes
  - Bubble : Echelle
  - High Low Open Close : Couleurs d'affichage
  - Candlestick : Affichage du volume
  - ...



# Tableaux croisés



# Tableau croisé

---

- Les **tableaux croisés** sont des tableaux spécifiques dans lesquels les nombres de lignes et de colonnes ne sont pas connus à l'avance.
- Par exemple, les ventes des différents produit par années :
  - Ni le nombre d'années, ni le nombre de produits ne sont connus au moment du design
- Ils sont utilisés pour afficher des données agrégées avec de multiple niveaux de groupements de colonnes et de lignes
- Les calculs courants sont des totaux, des pourcentages, des moyennes



# Tableau croisé

---

	2007	2008	2009
Fraises	45	21	39
Cerises	40	25	42
Pommes	35	36	38

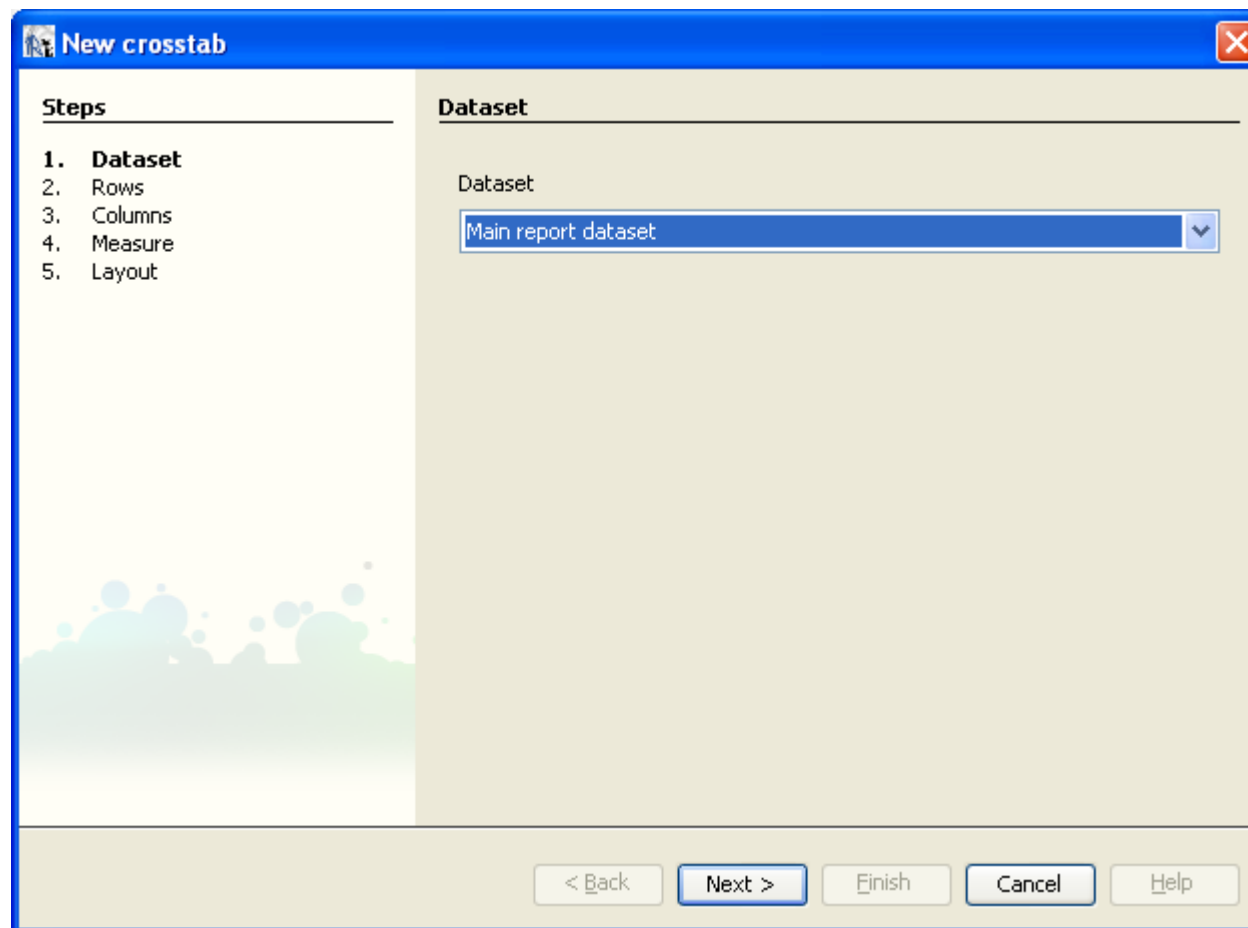


# Assistant

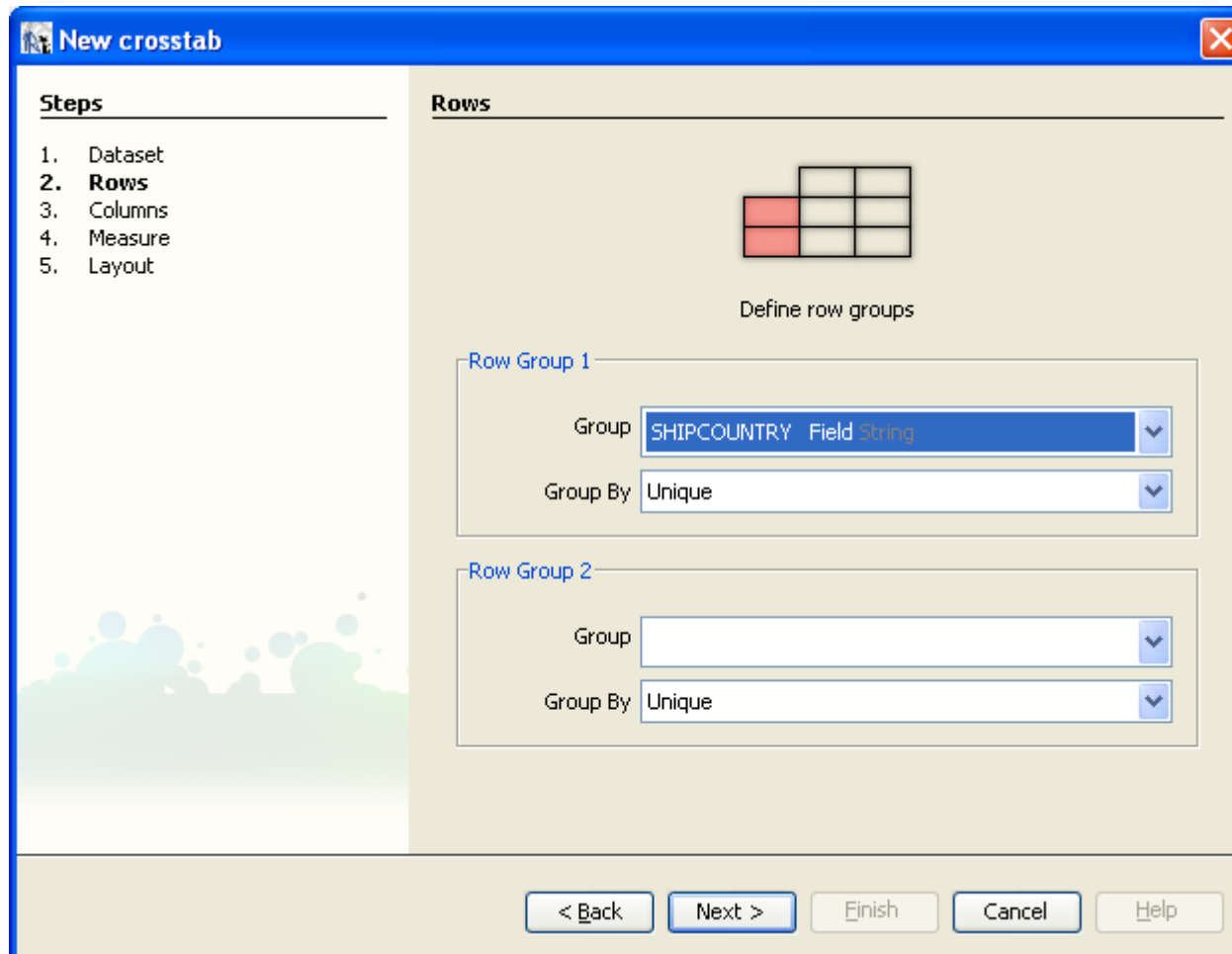
---

- La mise en place de tableau croisé est facilitée par un assistant en 5 étapes
  - Définition du dataset
  - Définition des regroupements de colonnes
  - Définition des regroupements de lignes
  - Définition de la mesure effectuée
  - Couleurs et ajout éventuels de totaux en début ou fin de ligne/colonne

# Etape 1 : Dataset



# Etape 2 : Lignes



The image shows a 'New Crosstab' dialog box with a blue title bar and a close button. On the left, a 'Steps' list shows: 1. Dataset, 2. Rows (highlighted), 3. Columns, 4. Measure, and 5. Layout. The main area is titled 'Rows' and contains a 3x2 grid icon with the text 'Define row groups' below it. There are two sections for row groups. 'Row Group 1' has a 'Group' dropdown set to 'SHIPCOUNTRY Field String' and a 'Group By' dropdown set to 'Unique'. 'Row Group 2' has empty 'Group' and 'Group By' dropdowns, both set to 'Unique'. At the bottom are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**New crosstab**

**Steps**

1. Dataset
2. **Rows**
3. Columns
4. Measure
5. Layout

**Rows**

Define row groups

Row Group 1

Group: SHIPCOUNTRY Field String

Group By: Unique

Row Group 2

Group:

Group By: Unique

< Back   Next >   Finish   Cancel   Help



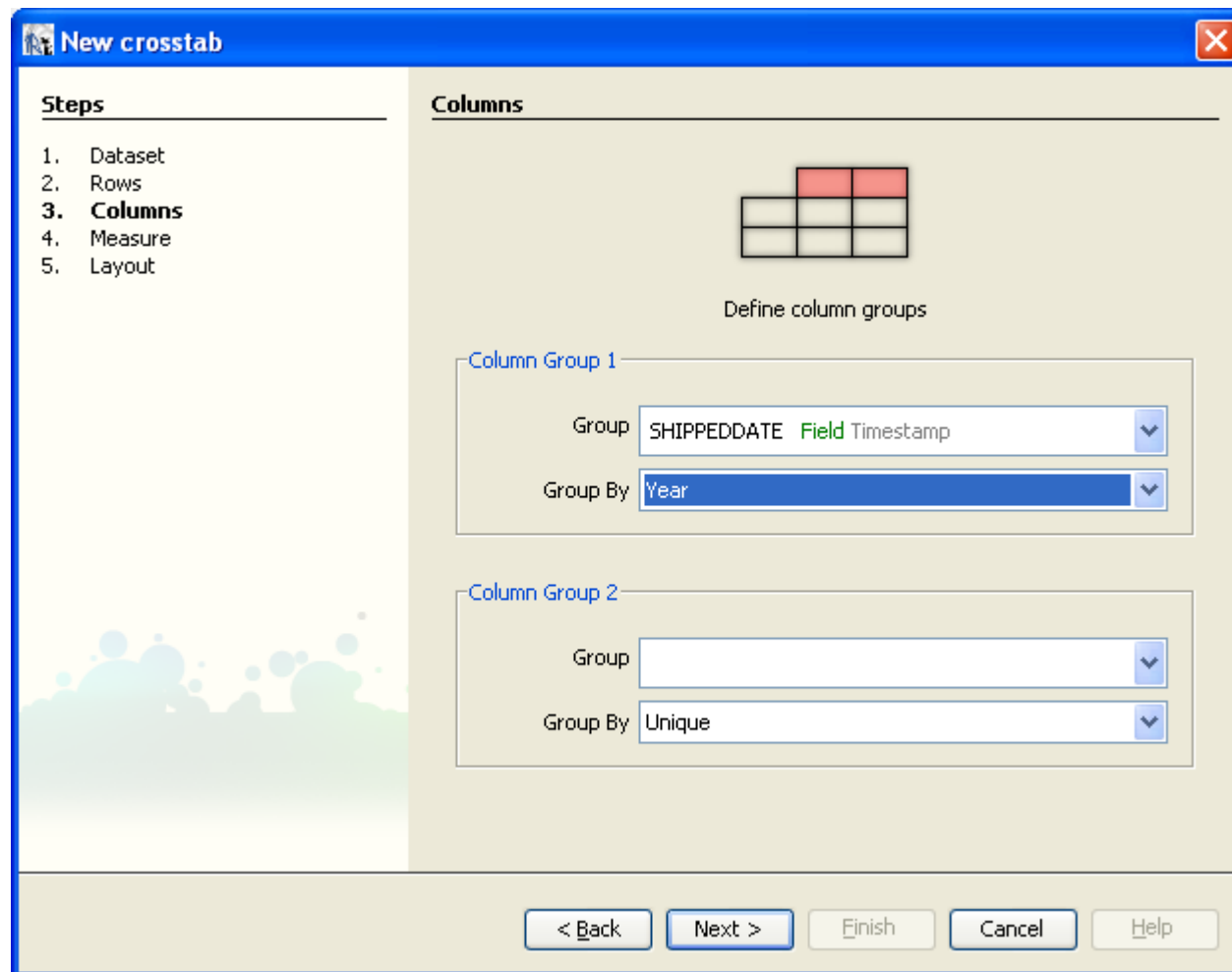
# Tri

---

- A la différence de l'utilisation des groupes dans le rapport principal, il n'est pas nécessaire de trier explicitement le jeu de données selon les critères de regroupement
  - Cela est fait automatiquement et peut être désactivé si les données sont déjà triées correctement



# Etape 3 : Colonnes



The image shows a screenshot of the 'New Crosstab' wizard, specifically the 'Columns' step. The window has a blue title bar with the text 'New crosstab' and a close button. On the left, a 'Steps' pane lists five steps: 1. Dataset, 2. Rows, 3. Columns (highlighted), 4. Measure, and 5. Layout. The main area is titled 'Columns' and contains a 3x2 grid of cells. The top-right cell is red, and the bottom-right cell is also red. Below the grid, the text 'Define column groups' is displayed. There are two sections for defining column groups. 'Column Group 1' has a 'Group' dropdown set to 'SHIPPEDDATE' with a green 'Field Timestamp' label, and a 'Group By' dropdown set to 'Year'. 'Column Group 2' has a 'Group' dropdown that is empty and a 'Group By' dropdown set to 'Unique'. At the bottom, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Steps**

1. Dataset
2. Rows
3. **Columns**
4. Measure
5. Layout

**Columns**

Define column groups

**Column Group 1**

Group: SHIPPEDDATE Field Timestamp

Group By: Year

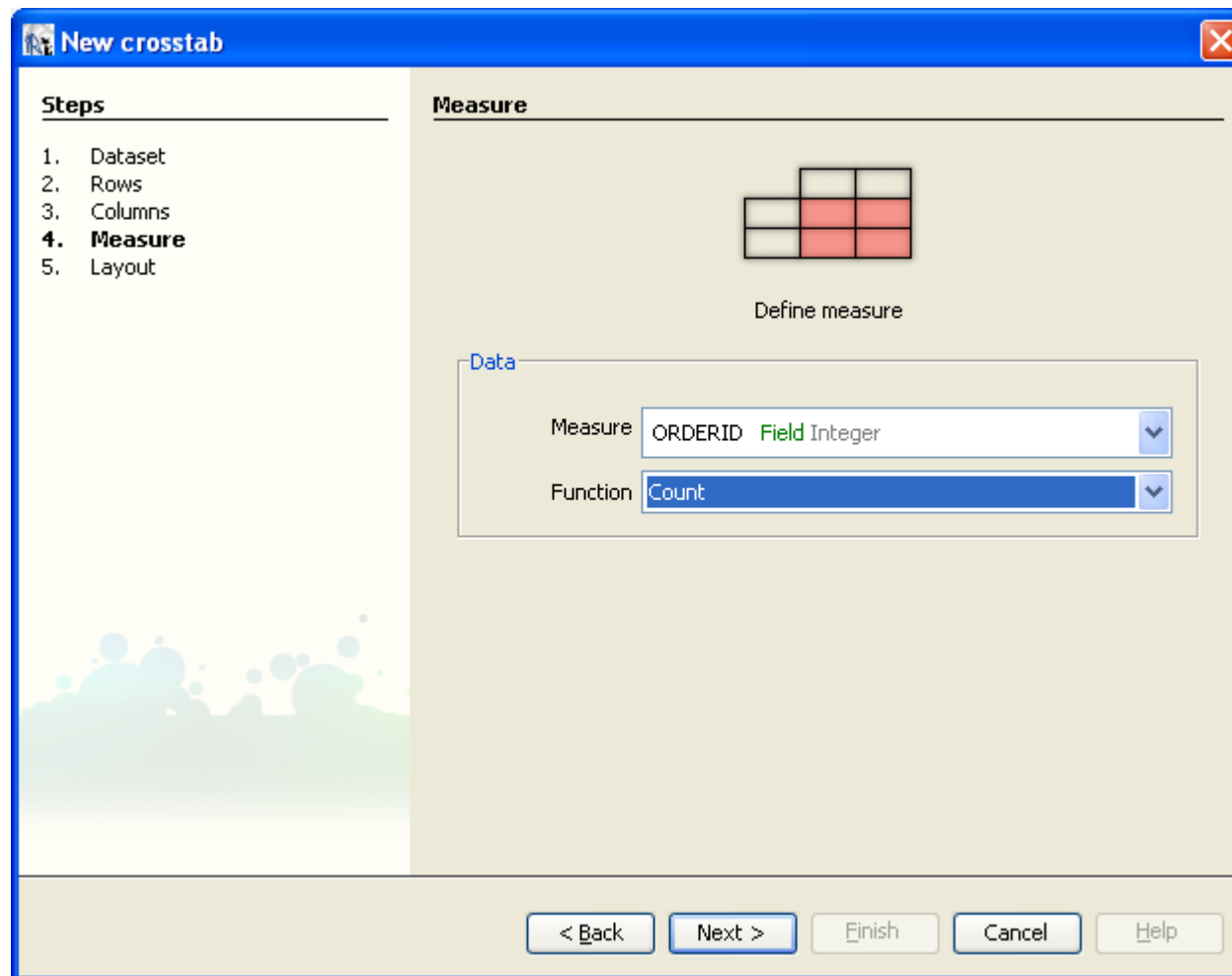
**Column Group 2**

Group:

Group By: Unique

< Back Next > Finish Cancel Help

# Etape 4 : Mesures



The image shows a 'New crosstab' dialog box with a blue title bar and a close button. It is divided into two main sections: 'Steps' on the left and 'Measure' on the right. The 'Steps' section contains a list of five steps: 1. Dataset, 2. Rows, 3. Columns, 4. Measure (which is highlighted in bold), and 5. Layout. The 'Measure' section features a 3x3 grid of cells, with the bottom-right cell highlighted in red. Below the grid is the text 'Define measure'. Underneath this is a 'Data' section containing two dropdown menus: 'Measure' set to 'ORDERID Field Integer' and 'Function' set to 'Count'. At the bottom of the dialog are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Steps**

1. Dataset
2. Rows
3. Columns
- 4. Measure**
5. Layout

**Measure**

Define measure

Data

Measure: ORDERID Field Integer

Function: Count

< Back Next > Finish Cancel Help



# Mesure

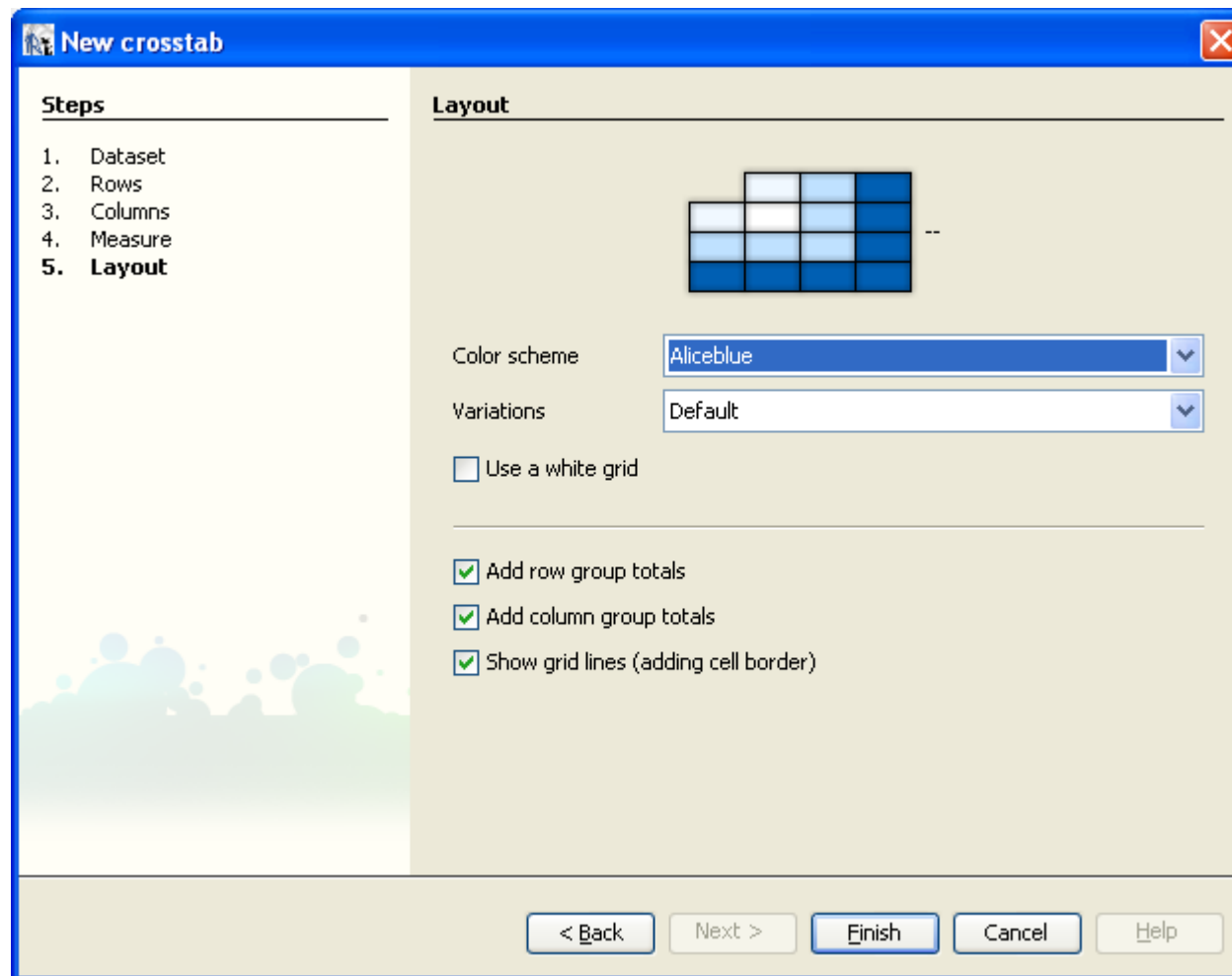
Une mesure est similaire à une variable. En général, c'est le résultat d'une fonction d'agrégation définie par :

- Son **nom**
- Sa **classe**
- Son **expression**
- Le type de calcul. Si les types de calcul proposés ne suffisent pas, les développeurs peuvent fournir une classe de type *Incrementer* par l'intermédiaire d'une factory

Exemple :

```
<measure name="ORDERIDMeasure"  
  class="java.lang.Integer" calculation="Count">  
  
  <measureExpression>  
    <![CDATA[$F{ORDERID}]]>  
  </measureExpression>  
  
</measure>
```

# Etape 5 : Totaux



The image shows a 'New crosstab' dialog box with a blue title bar and a close button. It is divided into two main sections: 'Steps' and 'Layout'. The 'Steps' section on the left lists five steps: 1. Dataset, 2. Rows, 3. Columns, 4. Measure, and 5. Layout, with the fifth step being highlighted. The 'Layout' section on the right features a 4x4 grid of colored squares (white, light blue, and dark blue) with a '--' symbol to its right. Below the grid are two dropdown menus: 'Color scheme' set to 'Aliceblue' and 'Variations' set to 'Default'. There are three checkboxes: 'Use a white grid' (unchecked), 'Add row group totals' (checked), 'Add column group totals' (checked), and 'Show grid lines (adding cell border)' (checked). At the bottom, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Steps**

1. Dataset
2. Rows
3. Columns
4. Measure
5. **Layout**

**Layout**

Color scheme: Aliceblue

Variations: Default

☐ Use a white grid

☒ Add row group totals

☒ Add column group totals

☒ Show grid lines (adding cell border)

< Back   Next >   Finish   Cancel   Help



# Cellules

---

- En fonction, du nombre de regroupement des lignes et des colonnes; le nombre de cellules du tableau croisé varie.
- On distingue :
  - Les cellules entête affichant des titres ou des noms de groupe
  - Les cellules affichant des totaux
  - Les cellules affichant les valeurs mesurées
- Du point de vue de *jasperStudio*, chaque cellule contient de éléments dont la valeur est définie par des expressions (Chaînes de caractères, variables, ...)



# Exemple

	1996	1997	1998	null	Total SHIPPED
Argentina	0	6	8	2	16
Austria	7	20	11	2	40
Belgium	2	7	10	0	19
Brazil	13	39	29	2	83
Canada	4	17	8	1	30
Denmark	2	11	4	1	18
Finland	4	13	5	0	22
France	15	38	22	2	77
Germany	23	60	37	2	122
Ireland	4	11	4	0	19
Italy	3	14	10	1	28
Mexico	9	12	6	1	28
Norway	1	2	3	0	6
Poland	1	2	4	0	7
Portugal	3	8	2	0	13
Spain	6	5	12	0	23
Sweden	6	17	14	0	37
Switzerland	3	8	6	1	18
UK	10	26	20	0	56
USA	20	62	37	3	122
Venezuela	7	20	16	3	46
Total	143	398	268	21	830

# Exemple 2 lignes/1 colonnes

		1996	1997	1998	null	Total SHIPPEDD
Portugal	Lisboa	3	8	2	0	13
Spain	Barcelona	1	2	2	0	5
	Madrid	4	1	3	0	8
	Sevilla	1	2	7	0	10
Sweden	Bräcke	3	7	9	0	19
	Luleå	3	10	5	0	18
Switzerland	Bern	2	3	3	0	8
	Genève	1	5	3	1	10
UK	Colchester	2	6	5	0	13
	Cowes	3	4	3	0	10
	London	5	16	12	0	33
USA	Albuquerque	6	6	5	1	18
	Anchorage	2	4	4	0	10
	Boise	1	19	11	0	31
	Butte	0	2	1	0	3
	Elgin	1	4	0	0	5



# Attributs d'un groupe

---

- ***totalPosition*** : Définit la présence d'une ligne (ou colonne) pour les totaux.
- ***order*** : L'ordre de tri dans le groupe (ascendant ou descendant)
- ***comparatorExpression*** : Retourne une instance de *java.util.Comparator* qui peut être utilisée pour trier les données





# Attributs tableau croisé

---

- ***isRepeatColumnHeaders*** : Les entêtes de colonnes sont répétées lors d'un changement de page
- ***isRepeatRowHeaders*** : Les entêtes de lignes sont répétées lors d'un changement de page
- ***columnBreakOffset***: L'espace entre deux parties du tableau croisé, lorsque celui-ci dépasse en largeur



# Variables du tableau croisé

---

- Un tableau croisé génère un **ensemble de variables**.
- Seuls ces variables peuvent être utilisées dans les expressions des cellules
- Les variables consistent en fait aux agrégations des mesures dans toutes les dimensions possibles du tableau.
- Elles sont utiles pour créer des **mesures dérivées**.



# Mesures dérivées

- Par exemple pour calculer un pourcentage, on peut diviser la mesure par la mesure agréée toutes dimensions confondues.
- L'expression serait alors:

```
new Double(  
    $V{ORDERIDMeasure}.doubleValue()  
    /  
    $V{ORDERIDMeasure_ORDERDATE_ALL.doubleValue() }  
)
```

- Et Le pattern : #,##0.00 %.

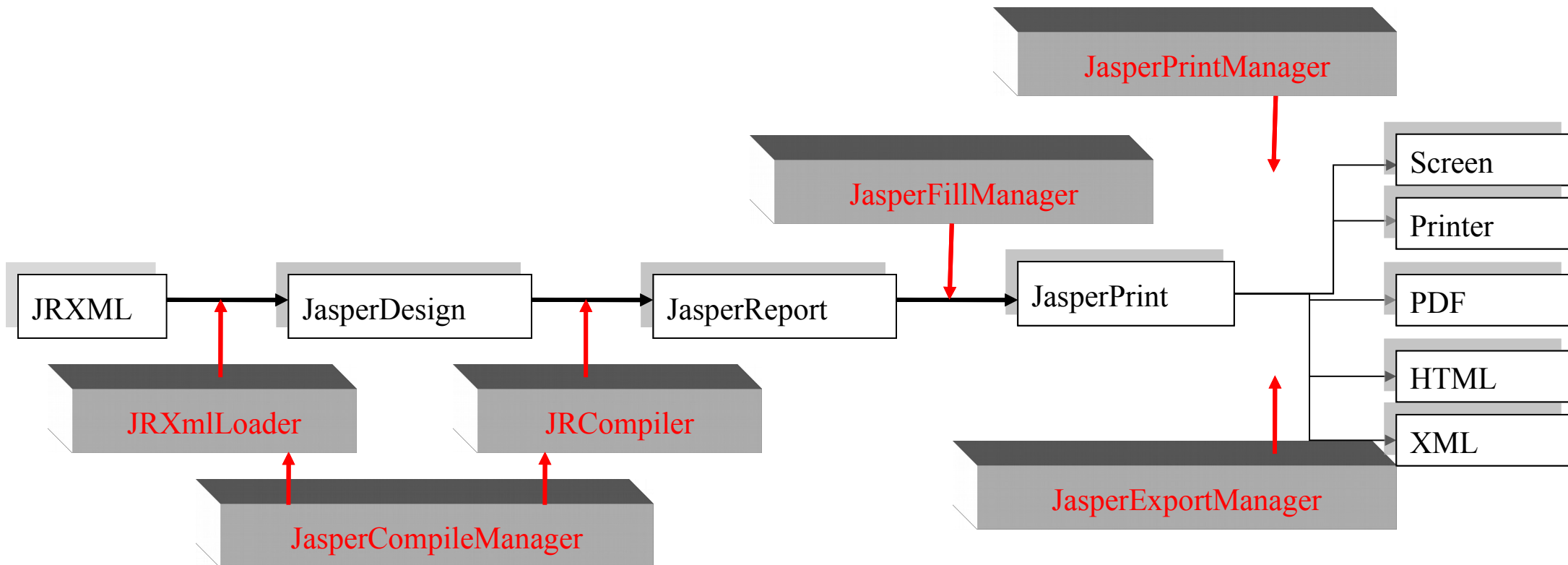


# Java et JasperReport

---

API  
Scriptlets  
Génération dynamique

# Cycle de vie d'un rapport





# Classes principales

La librairie JasperReport expose un ensemble de classes façades qui contiennent différentes méthodes statiques simplifiant l'accès à l'API.

Ces classes sont présentes dans le package *net.sf.jasperreports.engine* :

- **JRXmlLoader** : Charge un fichier JRXML
- **JasperCompileManager** : Compile les fichiers JRXML
- **JasperFillManager** : Fusionne les fichiers compilés avec les données dynamiques
- **JasperPrintManager** : Impression du document final
- **JasperExportManager** : Exportation du document final en PDF, HTML ou XML



# Chargement du fichier

La classe **JRXmlLoader** permet de construire un objet *JasperDesign* à partir d'un fichier XML

```
static JasperDesign load(java.io.File file)
static JasperDesign load(java.io.InputStream is)
static JasperDesign load(java.lang.String sourceFileName)
```

Exemple :

```
JasperDesign design = JRXmlLoader.load("/public/myReport.jrxml");
```

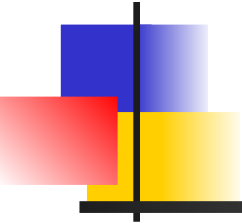


# Compilation

---

- La compilation du fichier JRXML est exécutée par les méthodes *compileReport()* de la classe ***JasperCompileManager***.
- Le résultat est un objet de type ***JasperReport***. Cet objet peut être sérialisé sur un disque pour se matérialiser en un fichier .jasper
- Le fichier .jasper peut faire partie du déploiement de l'application
- L'objet *JasperReport* ou le fichier .jasper sont utilisés lorsque l'application veut remplir la structure du rapport avec les données
- Il existe différents types de compilation selon que l'on utilise les langages Java, Groovy ou BeanShell





# Méthodes de *JasperCompileManager*

---

- *JasperCompileManager* propose des méthodes :
  - prenant en paramètre d'entrée un *InputStream*, un emplacement de fichier ou un objet *JasperDesign*
  - Retournant un *OutputStream*, un objet *JasperReport* ou générant un fichier *.jasper* en sortie

```
static JasperReport compileReport(java.io.InputStream inputStream)
static JasperReport compileReport(JasperDesign jasperDesign)
static JasperReport compileReport(java.lang.String sourceFileName)
static void compileReportToFile(JasperDesign jasperDesign, java.lang.String destFileName)
static String compileReportToFile(java.lang.String sourceFileName)
static void compileReportToFile(java.lang.String sourceFileName, java.lang.String destFileName)
static void compileReportToStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)
static void compileReportToStream(JasperDesign jasperDesign, java.io.OutputStream outputStream)
```



# Déploiement

---

Dans la majorité des cas, les fichiers JRXMLs ne changent pas à l'exécution de l'application finale, celle-ci ne fait que fournir dynamiquement les données et les paramètres à des rapports prédéfinis.

Dans ce cas, les fichiers JRXML peuvent être considérés comme du code source de l'application et leur compilation doit logiquement s'effectuer durant la construction de l'application

Les fichiers compilés *.jasper* sont alors inclus dans l'application déployée



# Compilateur JDT embarqué

Dans des cas plus complexes, les rapports peuvent être modifiés lors de l'exécution de l'application. Il est alors recommandé d'utiliser le compilateur JDT pour plusieurs raisons :

- Il n'a pas besoin d'utiliser de fichiers temporaires (le compilateur basé sur le JDK en a besoin).
- Il utilise le classloader de l'application pour résoudre les classes tandis que le compilateur JDK nécessite la notion de classpath.

L'utilisation du compilateur basé sur JDT nécessite de packager son jar avec la librairie de *JasperReports*. Cette librairie fait alors partie du déploiement



# Utilisation de l'outil Ant

---

- Comme la compilation s'effectue généralement lors du processus de build, une tâche Ant est fournie par JasperReport
- Cette tâche implémentée par la classe *JRAntCompileTask* est très similaire à la tâche prédéfini `<javac>`
- La définition de cette tâche dans un fichier ant *build.xml* s'effectue comme suit :

```
<taskdef name="jrc"
  classname="net.sf.jasperreports.ant.JRAntCompileTask">
  <classpath>
    <fileset dir="./lib"> <include name="**/*.jar"/> /fileset>
  </classpath>
</taskdef>
```

```
<jrc srcDir= »src/jrcml/* targetDir= »build/jasper »>
```



# Fusion de données

- La classe *JasperFillManager*
  - Prend en entrée un fichier compilé sous ses différents formats, un ensemble de paramètres, une source de données
  - Génère en sortie un objet *JasperPrint*, un fichier ou un objet *OutputStream* qui pourra être exporté en différents formats

```
static JasperPrint    fillReport(java.io.InputStream inputStream,  
                                java.util.Map parameters,  
                                java.sql.Connection connection)
```

```
static JasperPrint    fillReport(JasperReport jasperReport,  
                                java.util.Map parameters,  
                                JRDataSource dataSource)
```

```
static JasperPrint    fillReport(java.lang.String sourceFileName,  
                                java.util.Map parameters,  
                                JRDataSource dataSource)
```

- Exemple

```
JasperFillManager.fillReport(myReport, parametersMap, jdbcConnection);
```



# Exporters

---

- Les classes **exporters** permettent d'exporter un rapport généré vers un format spécifique de documents.  
Elles sont situées dans le package *net.sf.jasperreports.engine.export*
- La classe *JasperExportManager* permet d'exporter vers PDF, HTML et XML
- Pour les autres formats ou lors de besoins spécifiques, d'autres classes sont disponibles :
  - *JRHtmlExporter*, *JRPdfExporter* permettent de contrôler finement mapping des polices AWT vers les polices HTML ou PDF
  - *JRXlsExporter* vers Excel,
  - *JRCsvExporter* vers un format csv



# Méthodes de *JasperExportManager*

- La classe ***JasperExportManager*** :
  - Prend en entrée un objet *JasperPrint*, un *InputStream* ou un fichier
  - Génère un fichier, un *OutputStream*, un tableau d'octets dans un format de visualisation

```
static void    exportReportToHtmlFile(JasperPrint jasperPrint, java.lang.String destFileName)
static byte[]  exportReportToPdf(JasperPrint jasperPrint)
static void    exportReportToPdfFile(JasperPrint jasperPrint, java.lang.String destFileName)
static void    exportReportToPdfStream(java.io.InputStream inputStream, java.io.OutputStream
    outputStream)
static java.lang.String  exportReportToXml(JasperPrint jasperPrint)
static java.lang.String  exportReportToXmlFile(java.lang.String sourceFileName, boolean
    isEmbeddingImages)
```

- Exemple :  
`byte[] pdfContent = JasperExportManager.exportToPdf(myPrint);`



# Viewer

---

- JasperReport fournit plusieurs moyens pour visualiser directement un rapport généré. (*JasperPrint*)
  - A l'intérieur d'une application de type SWING, le composant de type *JPanel* : *JRViewer* peut être utilisé.
  - L'application stand-alone *JasperViewer* est une application Swing contenant le composant *JRViewer* (serialized JasperPrint objects) or in XML format.
- Au moment de la mise au point du rapport, JasperReport fournit également via la classe *JasperDesignViewer* la possibilité de prévisualiser un rapport.
- La plupart du temps les fichiers générés sont visualisés par les outils externes de l'utilisateur final





# Exemple servlet

---

```
public void service(HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException {
    response.setContentType("application/pdf");

    try {
        JasperReport report =
            JasperCompileManager.compileReport(request.getParameter("jrxml"));
        JasperPrint print =
            JasperFillManager.fillReport(report, request.getParameterMap(), connectio
            n);
        JasperExportManager.exportReportToPdfStream(print,
            response.getOutputStream());
    } catch (JRException e) {
        e.printStackTrace();
    }
}
```



# Scriptlets

- Les scriptlets offre une alternative aux expressions lorsque l'on nécessite des calculs plus complexes.
- Ce sont des séquences de code Java, exécutées à chaque fois qu'un **événement rapport** survient, permettant d'affecter des valeurs aux variables du rapport.
  - Ex : démarrage d'une nouvelle page, fin d'un groupe
- Puisque les scriptlets travaillent principalement avec les variables de rapport; il est important de contrôler exactement le moment où les scriptlets sont exécutés.  
JasperReports permet l'exécution de code spécifique Java **avant ou après l'initialisation** des variables.



# Scriptlets

- Un scriptlet est une classe qui étend ***JRAbstractScriptlet*** ou ***JRDefaultScriptlet***. Le nom de la classe doit être spécifiée dans l'attribut *scriptletClass* de l'élément *<jasperReport>*.
- La création d'une classe scriptlet requiert l'implémentation de certaines méthodes appelées par le moteur au moment opportun lors de la fusion de données:  
*beforeReportInit()*, *afterReportInit()*, *beforePageInit()*, *afterPageInit()*, *beforeGroupInit()*, *afterGroupInit()*, etc.
- Il est également possible de récupérer une instance de la classe scriptlet via le paramètre prédéfini *REPORT\_SCRIPTLET*. On peut alors dans le cadre d'une expression appelée une méthode spécifique de la classe.



# Ressources

---

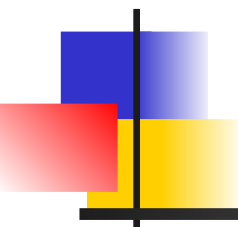
- ❖ “Ultimate Guides” fournis par JasperSoft (*JasperReport* et *iReport*)
- ❖ “JasperReports for Java Developer ”, David R. Heffelfinger, 2006
- ❖ Forums et Wikis
- ❖ <http://community.jaspersoft.com>



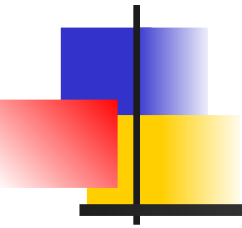
# Merci!!!

---

❖ MERCI DE VOTRE ATTENTION



# Annexes



# DataSources Jasper



# Interface JRDataSource

- L'interface *JRDataSource* définit deux méthodes :

- ***public boolean next()***

Cette méthode doit retourner *true* si le curseur est positionné correctement sur l'enregistrement suivant. Chaque fois que le moteur invoque cette méthode. Les valeurs des champs sont remplis avec les valeurs de l'enregistrement courant et les expressions sont réévaluées.

En conséquence, les entêtes d'un nouveau groupe peuvent être imprimées, un saut de page peut s'effectuer, ...

Si cette méthode retourne *false*, les bas de groupe, bas de colonne, la section de dernière page et le résumé sont imprimés.

- ***public Object getFieldValue(JRField jrField)***

Cette méthode retourne la valeur du champ passé en paramètre pour l'enregistrement courant.





# Collection de JavaBeans

---

- Un ensemble de JavaBeans implémente un *JRDataSource*
- Un JavaBean est une classe Java qui expose ses attributs avec des méthodes getter :  
*public <returnType> getXXX()*  
XXX est le nom du champ;
- Si le type de XXX est également un JavaBean. Il est possible d'utiliser la notation :  
XXX.YYY
- Le flag *Use Field description* permet d'utiliser la description du champ plutôt que son nom.
- La collection ou le tableau de JavaBeans doit être retourné par une méthode *static* d'une classe fournie (factory).



# Exemple JavaBean

---

```
public class PersonBean {  
    private String name = "";  
    private int age = 0;  
  
    public PersonBean(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
  
    public int getAge() {  
        return age;  
    }  
    public String getName(){  
        return name;  
    }  
}
```



# Exemple Factory

---

```
public class SampleFactory {  
  
    public static java.util.Collection generateCollection() {  
        java.util.Vector collection = new java.util.Vector();  
        collection.add(new PersonBean("Ted", 20) );  
        collection.add(new PersonBean("Jack", 34) );  
        collection.add(new PersonBean("Bob", 56) );  
        collection.add(new PersonBean("Alice",12) );  
        collection.add(new PersonBean("Robin",22) );  
        collection.add(new PersonBean("Peter",28) );  
        return collection;  
    }  
}
```



# XML DataSource

---

- A la différence du format « table » voulu par Jasper, un document XML est organisé hiérarchiquement en forme d'arbre
- Une expression *XPath* est alors nécessaire pour extraire un ensemble de nœuds du document.
- L'expression *XPath* peut être fournie :
  - dans le rapport; dans ce cas il est possible d'utiliser les paramètres du rapport dans l'expression *XPath*
  - Au moment de la création de la datasource.
- Au moment de la création du datasource, il est également possible de spécifier des patterns Java pour convertir les chaînes de caractères XML en dates ou nombres



# Champs XML

- La syntaxe *XPath* est également utilisée pour définir les champs. L'expression étant évaluée à partir du nœud courant. Elle se spécifie dans la description du champ.
- *IReport* propose un outil visuel pour la sélection des champs

Nom	Description	
id	@id	L'attribut id
lastname	lastname	La valeur du noeud-enfant lastname
categoryName	ancestor::category/ @name	L'attribut name du noeud ancêtre category



# Documents XML et sous-rapport

- L'organisation hiérarchique des documents est très adaptée à la notion de sous-rapport de JasperReport
- Une *JRXmlDataSource* expose 2 méthodes supplémentaires :
  - *public JRXmlDataSource dataSource(String selectExpression)*
  - *public JRXmlDataSource subDataSource(String selectExpression)*

La première méthode applique une expression *XPath* à partir du nœud racine du document XML. La seconde applique l'expression sur le nœud courant.

L'expression pour spécifier les données d'un sous rapport est alors :

```
((JRXmlDataSource)  
$P{REPORT_DATA_SOURCE}).subDataSource(xExpression)
```



# CSV DataSource

---

- Les fichiers CSV sont un format naturel pour une source de données Jasper.
- Les noms des champs sont :
  - Soit obtenus à partir de la première ligne du fichier
  - Soit spécifié un à un
- Il est possible également de spécifier le caractère délimiteur



# *JREmptyDataSource*

---

- JasperReport fournit une datasource spéciale permettant d'effectuer des itérations sur des enregistrements dont la valeur de champ est *null*
- L'interface iReport permet de spécifier le nombre d'itérations voulues





# Hibernate

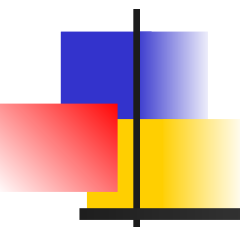
- Avec Jasper, il est possible d'utiliser une requête HQL.
- La connexion hibernate (*hibernate.cfg.xml*) et les fichiers de mapping (*\*.hbm.xml*) doivent être dans le classpath.
- La requête est exprimée alors via HQL.
- En fonction du résultat retourné par la requête, les champs du rapport sont alors des entités Hibernate ou des objets simples Java.
- Il est possible de naviguer dans les objets entités avec la notation « . »



# Datasources personnalisées

---

- Fournir une classe implémentant l'interface *JRDataSource*
- Fournir une factory ayant une méthode statique retournant la datasource personnalisée



# Templates et assistant



# Template

- Avec *JasperStudio*, il est possible de définir des **gabarits de rapport** (template) permettant la réutilisation de disposition et de présentation
  - Ces gabarits peuvent être associés à des assistants pour faciliter alors la création de rapport.
  - *JasperStudio* proposent quelques gabarits dans sa distribution.
- Les fichiers gabarits sont de simples fichiers JRXML stockés dans le répertoire de son choix  
*Windows → Preferences → Resource Folder → Template location*
- Lorsqu'un nouveau rapport est créé en utilisant un template, l'utilisateur a le choix entre :
  - Démarrer l'assistant associé au template qui lui permettra de renseigner les informations prévues
  - D'utiliser directement le fichier JRXML du template



# Assistant

---

- L'assistant est capable de créer à partir d'une liste de champs sélectionnés par l'utilisateur d'ajouter des éléments textes permettant d'afficher les enregistrements.
- Ces derniers peuvent également être groupés.
- Deux types de disposition sont disponibles :
  - **Colonnes** : Pour chaque enregistrement, on obtient deux colonnes pour le label du champ et sa valeur
  - **Tabulaire** (défaut) : Les enregistrements sont affichés en tableau, en utilisant les column header comme titre de colonne



# Colonne

---

## Classic template

---

Last name	Nowmer
First name	Sheri
Address	2433 Bailey Road
City	Tlaxiaco
Postal Code	15057
Country	Mexico

---

Last name	Whelply
First name	Derrick
Address	2219 Dewing Avenue
City	Sooke
Postal Code	17172
Country	Canada

---

Last name	Derry
First name	Jeanne
Address	7640 First Ave.
City	Issaquah
Postal Code	73980
Country	USA

---

# Tableau

## Classic template

Last name	First name	Address	City	Postal Code	Country
Nowmer	Sheri	2433 Bailey	Tlaxiaco	15057	Mexico
Whelply	Derrick	2219 Dewing	Sooke	17172	Canada
Derry	Jeanne	7640 First Ave.	Issaquah	73980	USA
Spence	Michael	337 Tosca Way	Burnaby	74674	Canada
Gutierrez	Maya	8668 Via Neruda	Novato	57355	USA
Damstra	Robert	1619 Stillman	Lynnwood	90792	USA
Kanagaki	Rebecca	2860 D Mt. Hood	Tlaxiaco	13343	Mexico
Brunner	Kim	6064 Brodia	San Andres	12942	Mexico
Blumberg	Brenda	7560 Trees	Richmond	17256	Canada
Stanz	Darren	1019 Kenwal Rd.	Lake Oswego	82017	USA
Murraiin	Jonathan	5423 Camby Rd.	La Mesa	35890	USA
Creek	Jewel	1792 Belmont	Chula Vista	40520	USA
Medina	Peggy	3796 Keller	Mexico City	59554	Mexico
Rutledge	Bryan	3074 Ardith	Lincoln Acres	30346	USA
Cavestany	Walter	7987 Seawind	Oak Bay	15542	Canada
Planck	Peggy	4864 San Carlos	Camacho	77787	Mexico
Marshall	Brenda	2687 Ridge	Arcadia	28530	USA
Wolter	Daniel	2473 Orchard	Altadena	49680	USA
Collins	Dianne		Oakland	21	USA
					USA



# Groupes

---

- L'utilisateur peut choisir de grouper les données. (4 groupes au maximum)
  - => Les entêtes et bas de groupe sont alors créés pour chaque groupe
  - => Pour chaque groupe, l'assistant positionne l'expression et ajoute un label et un champ texte permettant d'afficher la valeur.
- L'expression est obligatoirement réduite à un nom de champ





# Mise au point

---

- Pour mettre un point un gabarit compatible avec l'assistant. Il est nécessaire de respecter certaines **règles de nommage**.
- Le gabarit définit à priori :
  - Le maximum de groupe possible (4)
  - L'entête de colonne
  - La bande de détail
- Ces bandes sont analysées par l'assistant afin de remplacer ou de créer certains éléments textes.
- Les autres bandes ne sont pas modifiées par l'assistant



# Groupes

---

- Les éléments des entêtes de groupes sont analysés et si une de leur valeur correspond à une valeur prédéfinie, elle est remplacée soit par le nom du critère de groupe (Textes statiques) soit par la valeur de l'expression (Champs textes)
  - Textes statiques : *GroupLabel*, *Group Label*, *Label*, *Group name*, *GnLabel* (avec n le n° de groupe)
  - Champs textes : *GroupField*, *Group Field*, *Field*, *GnField*



# Colonne

---

- L'entête de colonne est analysée par l'assistant lors d'une disposition en table.
- Si il trouve des textes statiques contenant des chaînes prédéfinies, l'assistant créera un label pour chaque champ.
- Les valeurs prédéfinies étant :
  - *DetailLabel, Label* ou *Header*



# Détail

---

- Si le rapport à une disposition en table, l'assistant analysera la section détail afin de rechercher les champs textes contenant des valeurs prédéfinies.
- Il créera alors un champs texte pour chaque champs des enregistrements. Les valeurs prédéfinies étant :
  - “*DetailField*”, “*Field*”
- Si le rapport est en colonne, l'assistant recherchera dans la bande détail, les champs statiques avec les mêmes critères que l'entête de colonne.



# Type de disposition

---

- Pour choisir entre la disposition colonne et la disposition table, il suffit de spécifier dans le gabarit la propriété ***template.type*** qui peut prendre 2 valeurs :
  - *tabular*
  - *columnar*



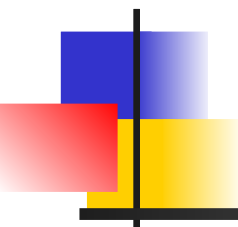
# Exemple *cherry.jrxml*

```
<jasperReport name="Cherry" language="groovy" pageWidth="595" pageHeight="842" columnWidth="535"
  leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
  <style name="Row" mode="Transparent" fontName="Times New Roman" pdfFontName="Times-Roman">
    <conditionalStyle><conditionExpression><![CDATA[$V{REPORT_COUNT}%2 == 0]]></conditionExpression>
    <style style="Row" mode="Opaque" bgcolor="#F0EFEF"/>
  </conditionalStyle></style>
  <group name="Group1">
    <groupExpression><![CDATA[(int) ($V{REPORT_COUNT}/15)]]></groupExpression>
    <groupHeader>
      <band height="37">
        <frame><reportElement mode="Opaque" x="0" y="7" width="555" height="24" forecolor="#B89F7D"
          bgcolor="#000000"/>
        <textField isStretchWithOverflow="true">
          <reportElement style="SubTitle" isPrintRepeatedValues="false" x="2" y="0" width="479" height="24"
            forecolor="FFFFFF"/>
          <textFieldExpression class="java.lang.String"><![CDATA["GroupField"]]></textFieldExpression>
        </textField>
      </frame>
    </band>
  </groupHeader>
  ...
</jasperReport>
```



# TP

- Création et utilisation d'un gabarit



# Report Books





# Introduction

---

Un **report book** est un unique fichier *.jrxml* qui englobe plusieurs rapports dans une unique objet.

Le livre a ses paramètres, ses variables, sa requête et ses champs

Le livre introduit la notion de **master** qui correspond au niveau livre :

- MASTER\_PAGE : Le numéro de la « partie » du livre
- Réinitialisation MASTER : Réinitialisation d'une variable à un changement de « partie »

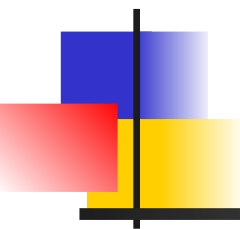


# Apports

---

Les report book permettent de :

- Générer un sommaire avec les différentes signets des parties du livre (i.e les sous-rapports associés)
- Paginiation spécifique incluant le n° de la partie



# Génération dynamique de rapport



# Introduction

---

- Dans certains cas, l'application Java finale peut vouloir mettre au point un design Jasper à la volée.
- L'API de JasperReport permet alors toutes les opérations sur une instance de *JasperDesign* :
  - Création
  - Ajout de champs, variables, groupes
  - Manipulation de bandes, sections
  - Ajouts d'éléments



# Cas d'utilisation

---

- Il est, cependant fastidieux de créer un *JasperDesign* complet à partir de zéro.
- En général, on part d'un fichier JRXML incomplet que l'on complète avec les informations récoltées de l'interface utilisateur. Par exemple, la requête les champs, les groupes, etc..



# Interface et classe d'implémentation

---

- L'API fournit deux types d'objets Java
  - Des **interfaces** qui permettent d'accéder en lecture aux éléments du rapport.  
Exemple, *JRGroup*, *JRBand*, *JRField*
  - Des **implémentations** qui peuvent elles être instanciées et modifiées  
Exemple : *JRDesignGroup*, *JRDesignBand*, *JRDesignField*
- Les méthodes disponibles renvoient des interfaces qu'il est nécessaire de *caster* afin de pouvoir les manipuler



# Hiérarchie

---

- La classe *JasperDesign* hérite de *JRBaseReport*.
  - ***JRBaseReport*** encapsule toutes les données concernant la structure du rapport (*JRSection* et *JRBand*)
  - ***JasperDesign*** complète la classe parente avec tous les concepts extérieurs à la structure : champs, variables, groupes, scriptlets, styles, ...



# *JRBaseReport*

- *JRBaseReport* permet d'accéder aux éléments structurels du rapport :
  - ***JRSection*** : Une section peut contenir plusieurs bandes. Il existe trois types de sections dans un rapport
    - La section détail : *JRSection getDetailSection()*
    - Les sections relatives aux groupe : *groupHeaderSection* et *groupFooterSection* obtenues à partir de la classe *JGroup*
  - ***JRBand*** : Une bande contenant des éléments
    - *JRBand getTitle();*
    - *JRBand getBackground();*
    - *JRBand getColumnHeader() , getColumnFooter()*
    - *JRBand getPageHeader(), getPageFooter(), getLastPageFooter()*
    - *JRBand getSummary()*
    - *JRBand getNoData()*





# *JRDesignBand*

- A partir d'une instance de *JRDesignBand*, il est possible d'ajouter des éléments dans une bande du rapport via les méthodes :
  - *addElement(JRDesignElement element)*
  - *addElementGroup(JRDesignElementGroup group)*
- ***JRDesignElement*** est un élément simple qui ne contient pas d'autre élément
- ***JRDesignElementGroup*** est un container d'éléments (Ex *JRDesignBand*)



# *JRDesignElement*

---

- Les classes implémentant *JRDesignElement* correspondent à tous les éléments qu'il est possible d'ajouter dans une bande :
  - *JRDesignTextElement*, *JRDesignCrosstab*,  
*JRDesignGraphicElement*, *JRDesignChart*,  
*JRDesignFrame*, *JRDesignSubreport*, ...
- Sur chaque élément, il est possible d'accéder à ses propriétés spécifiques (police, radius, dataset, etc.)



# Méthodes de lecture JasperDesign

- En dehors de la structure, des méthodes de type *getter* permettent de récupérer toutes les autres propriétés du design :
  - Toutes les expressions utilisées :
    - `Collection getExpressions()`, `JRExpression getFilterExpression()`
  - Les champs :
    - `List getFieldsList()`, `Map getFieldsMap()`
  - Les paramètres :
    - `List getParametersList()`, `Map getParametersMap()`
  - Les variables :
    - `List getVariablesList()`, `Map getVariablesMap()`
  - Les groupes :
    - `List getGroupsList()`, `Map getGroupsMap()`



# Méthodes de lecture JasperDesign

---

- Les scriptlets :
  - `List getScriptletsList(), Map getScriptletsMap()`
- Les styles
  - `List getStylesList(), Map getStylesMap()`
- Les tableaux croisés :
  - `List getCrosstabs()`
- Les datasets :
  - `JRDesignDataset getMainDesignDataset(), Map getDatasetMap(), JRDataset[] getDatasets(), List getDatasetsList()`



# Méthode d'ajout d'éléments

---

- Méthodes d'ajouts :
  - `void addField(JRField field)`
  - `void addParameter(JRParameter parameter)`
  - `void addGroup(JRDesignGroup group)`
  - `void addVariable(JRDesignVariable variable)`
  - `void addScriptlet(JRScriptlet scriptlet)`
  - `void addStyle(JRStyle style)`
- Les méthodes *remove* équivalentes sont bien évidemment accessibles
- La méthode pour spécifier la requête :  
`setQuery(JRDesignQuery query)`



# Expressions

---

- En dehors de leurs attributs spécifiques, les éléments ont en général besoin de définir leur expression et la classe associée.
- La classe peut être précisée soit par une *String* (*classname*), soit directement par une instance de *java.lang.Class*
- L'expression est elle fournie par une instance de ***JRExpression***



# Composition d'une expression

---

- Une expression est découpée en morceau (***JRChunkExpression***)
- Chaque morceau étant constitué d'une chaîne de caractère et d'un type.
- Les types possibles étant :
  - *JRExpressionChunk.TYPE\_TEXT*
  - *JRExpressionChunk.TYPE\_PARAMETER*
  - *JRExpressionChunk.TYPE\_FIELD*
  - *JRExpressionChunk.TYPE\_VARIABLE*
  - *JRExpressionChunk.TYPE\_RESOURCE*



# TP

---

- Compléter un gabarit JRXML  
programmatically