

Restitution

Projet Darties – Groupe2

utilisation jasper --> java

**Création du document : GORGONE Stéphanie – MANDON Chloé**

**Date : 15/12/2010**

utilisation jasper --> java

Projet Darties – Groupe2

[I. Introduction 2](#_Toc280177882)

[II. Lien entre un rapport Jasper et une application Java 3](#_Toc280177883)

[III. Requêtes à la volée 6](#_Toc280177884)

[1. A l'aide d'un paramètre 6](#_Toc280177885)

[2. A l'aide d'objets Java : JRDataset et JRDesignQuery 7](#_Toc280177886)

[IV. Modification du fichier xml 8](#_Toc280177887)

[1. Rappel sur JASPERREPORT 8](#_Toc280177888)

[2. Utilisation de parseur 9](#_Toc280177889)

[Présentation 9](#_Toc280177890)

[Architecture 9](#_Toc280177891)

[V. Conclusion 10](#_Toc280177892)

# Introduction

Ce rapport intervient dans le cadre du développement du projet décisionnel DARTIES par des étudiants en dernière année d'école d'ingénieur en Informatique (MIAGE). Il traite plus précisément de la partie "Restitution - Jasper" du projet réalisé par le groupe 2.

Ce rapport va permettre, tout d'abord, d'expliquer le lien entre un rapport Jasper et une application Java, c'est-à-dire les différentes étapes et objets permettant d'afficher le rapport en Java.

Ensuite, trois méthodes envisageables pour effectuer des requêtes à la volée vont être définies et brièvement comparées.

Ainsi, ce dossier devrait permettre à toute personne du groupe de se lancer assez aisément dans la restitution du projet.

# Lien entre un rapport Jasper et une application Java

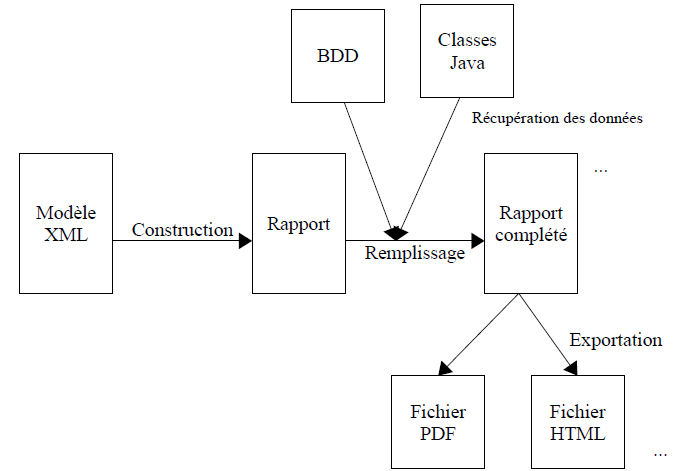
Lorsque l'on parle d'un rapport Jasper, on parle de la page qui contient le tableau et/ou le graphique à afficher dans notre application Java.

Un rapport Jasper est réalisé à l'aide d'un éditeur graphique, iReport en ce qui nous concerne. Cela permet le paramétrage et la mise en place aisés du rapport.

Ce dernier est ensuite enregistré sous forme d'un document XML (extension "jrxml").

La création de rapports en java se déroule donc généralement en 4 étapes :

* l'obtention d'un fichier modèle XML à l'aide d'éditeurs graphiques
* la construction du rapport à partir du modèle
* le remplissage des différents champs du rapport avec les données en provenance de diverses sources (bases de données, classes Java, ...)
* l'exportation du résultat dans plusieurs formats possibles (PDF, HTML, ...)

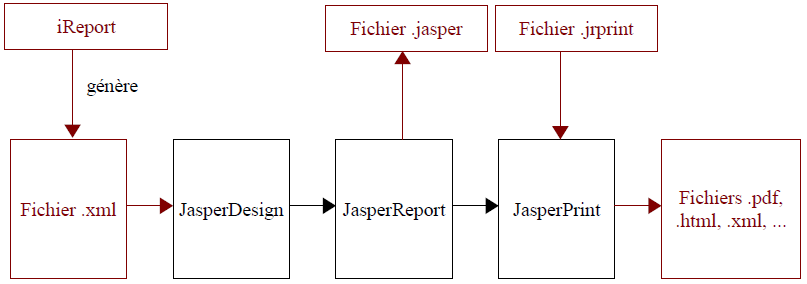


Afin de parvenir à l'affichage du rapport en Java, il faut passer par différents objets Java.

En effet, une fois le modèle XML (objet JasperDesign) compilé, il est chargé dans un objet Java (JasperReport) qui peut lui-même être sérialisé et stocké dans un fichier (avec l'extension .jasper). Cet objet sérialisé est alors utilisé lorsque l'application désire compléter le rapport avec des données.

Le résultat obtenu après le processus de remplissage des champs est un nouvel objet Java (JasperPrint) qui représente le document final.

Celui-ci peut être soit stocké sur disque pour un usage ultérieur (sous forme sérialisée et avec l'extension .jrprint), soit directement imprimé ou encore transformé dans un format lisible (PDF, HTML, ...).

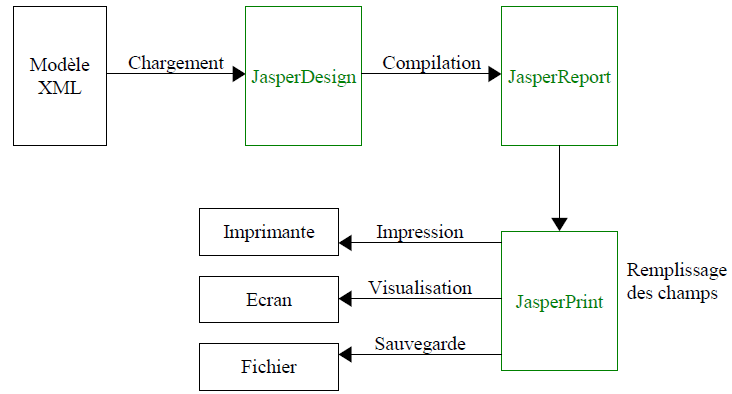


Evidemment, le document XML peut être créé sans iReport en respectant la structure du fichier, mais cet outil est tout de même plus pratique. Il est vrai que pour certaines modifications, il est utile de pouvoir agir directement sur le document XML.

Dans ce document sont définis entre autres : les champs provenant d'une source de données, des paramètres, des variables, des expressions, des requêtes, etc.

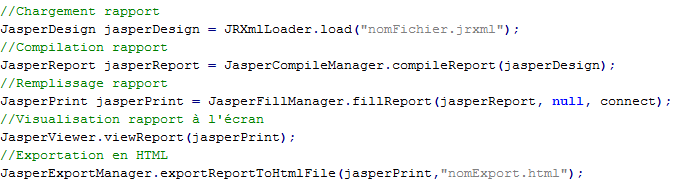
Avant de présenter un exemple de code Java, nous allons succinctement décrire les différents objets énumérés ci-dessus :

* **JasperDesign** : représente la définition d'un rapport. Il faut créer un objet JasperDesign à partir d'un fichier modèle XML.
* **JasperReport** : représente un objet JasperDesign compilé. Le processus de compilation vérifie la structure du modèle XML, le compile et le stocke dans un objet JasperReport.
* **JasperPrint** : représente le rapport final. Un objet JasperPrint est élaboré à partir d'un objet JasperReport par un processus de remplissage qui consiste à insérer dans le rapport des données en provenance d'une source de données quelconque.



Un certain nombre de librairies sont nécessaires et doivent être ajoutées au projet Java, sans quoi ces objets ne seraient pas utilisables :

* **jasperreports.jar** : toutes les fonctionnalités propres à JasperReports.
* **commons-digester.jar** : tranformer un fichier XML en objets Java.
* **commons-collections.jar** : nouvelles classes, interfaces fonctionnalités supplémentaires aux classes de base (en particulier l'interface Map).
* **commons-beanutils.jar** : aide à la génération d'objets Java.
* **commons-logging.jar** : fournit, de manière transparente, des fonctionnalités de log en utilisant n'importe lequel des frameworks de log existants.
* **jdt-compiler.jar** : obligatoire pour développer avec Eclipse.
* **ojdbc14.jar** : communiquer avec la base de données (Oracle dans notre cas).

Voici alors le code Java nécessaire à l'affichage d'un rapport en partant d'un modèle XML. La connexion à la base de données doit avoir été préalablement effectuée et l'objet "connect" correspond à un objet java.sql.Connection.

Le deuxième paramètre de la fonction "fillReport" correspond à une table HashMap, qui doit être antérieurement créée, et qui contient des paramètres à fournir éventuellement au rapport. Nous n'utilisons pas de paramètre ici, d'où le "null", mais nous allons voir comment les utiliser dans la partie suivante.

# Requêtes à la volée

Un objectif primordial de la partie "Restitution" du projet est de parvenir à changer la requête du tableau "à la volée". En effet l'utilisateur doit pouvoir changer un certain nombre de paramètres à afficher dans le tableau chaque fois qu'il le souhaite.

Pour cela nous avons identifié deux méthodes permettant de changer la requête sur laquelle est basé le tableau.

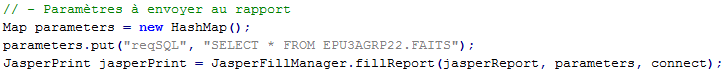
Nous rappelons qu'un tableau est créé à partir d'un dataset. C'est ce dataset qui contient la requête. Puis le nom des champs dans les colonnes du tableau correspondent au nom des champs du select de la requête.

## A l'aide d'un paramètre

Nous avons parlé de la possibilité d'associer des paramètres à un rapport Jasper, ces paramètres étant fournis au moment du remplissage du rapport.

Il est donc possible d'ajouter un paramètre de type String au dataset du tableau, vide dans un premier temps. Ensuite, dans le modèle XML on donne le nom du paramètre à la place de la requête du dataset.

Ainsi, avant d'appeler la méthode de remplissage, on crée une HashMap, on lui ajoute la correspondance entre le nom du paramètre créé et la requête que l'on veut effectuer, et on fournit la table à la méthode. Voici le code :



Le paramètre "reqSQL" doit bien être auparavant ajouté au dataset dans le modèle XML. Puis la méthode de remplissage associe la requête au paramètre et le remplace ainsi dans l'emplacement de la requête du dataset.

Or, il s'est avéré, sauf erreur de notre part, qu'il soit difficile de remodifier le contenu d'un paramètre après qu'il lui ait été affecté.

Nous n'avons donc pas réussi à affecter une autre requête à ce même paramètre dans la même exécution, son contenu restait inchangé.

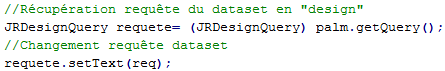
## A l'aide d'objets Java : JRDataset et JRDesignQuery

L'objet JasperDesign décrit antérieurement et créé à partir du modèle du rapport permet ensuite d'accéder aux différents éléments du rapport, notamment à la liste des datasets, comme suit :



Nous avons ensuite récupéré l'objet JRDataset à l'indice 0 car nous savons qu'il y en aura toujours un seul par rapport dans notre cas.

Cet objet nous permet d'accéder à la requête du dataset, comme suit :



La requête est récupérée dans un objet JRDesignQuery et non pas JRQuery (d'où le cast), car nous avons besoin de la méthode "setText()" pour changer la requête, et cette dernière n'est pas fournie par l'objet JRQuery.

L'objet "req" dans le code ci-dessus correspond à un objet String créé au préalable contenant la requête à associer au tableau.

Il faut ensuite recompiler le rapport, le remplir et l'exporter comme nous l'avons montré précédemment.

Cette méthode permet bien de changer la requête autant de fois qu'on le désire dans une même exécution basée sur un seul modèle XML.

Cependant, ces méthodes fonctionnent dans le cas où seul le contenu du tableau change, suite à des restrictions supplémentaires dans la requête, mais la structure du tableau ne doit pas changer.

En effet, les champs récupérés dans le select de la requête doivent être impérativement les mêmes car ils correspondent aux colonnes du tableau. Le nombre de colonnes ne doit d'ailleurs pas changer non plus.

Si l'utilisateur affichait un tableau de chiffre d'affaires (objectif, réalisé, écart), et qu'il souhaite ensuite afficher le même tableau mais pour la marge brute, cette méthode ne suffit pas.

Il y a alors deux solutions : soit on crée un modèle XML pour chaque structure de tableau possible et on utilise cette méthode, soit on utilise les parseurs pour agir sur le modèle XML.

Cette dernière méthode va donc être décrite dans la partie suivante.

# Modification du fichier xml

## Rappel sur JASPERREPORT

Un rapport Jasper permet de faire une synthèse de données sous forme de tableaux, graphiques…

D’un point de vue technique, un rapport Jasper passe par deux étapes successives. Dans un premier temps, un fichier XML (.jrxml) est généré via le logiciel fourni par Jasper afin d’éditer des rapports plus facilement : IReport. Dans un second temps, ce fichier devra être compilé en un fichier .jasper pour être lu par le serveur d’application.

Cet outil open-source permet de générer plusieurs types de fichiers résultats comme :

* PDF
* XLS (Excel)
* ODF (OpenOffice)
* RTF (Word en autres)
* HTML
* …



.jasper

.jrxml

## Utilisation de parseur

### Présentation

Il est possible de gérer un rapport Jasper sans utiliser le logiciel graphique IReport. En effet, on dispose d’une bibliothèque de fonctions JAVA permettant de manipuler un rapport Jasper.

A l’aide du langage JAVA, il peut être parfois compliqué d’effectuer certaines modifications dans un tableau. C’est pour cela, que l’on a décidé de transformer le fichier source XML (.jrxml) via un « parseur » de flux. Ce parseur va permettre de travailler avec les balises XML et ainsi de simplifier la modification de tableaux. En effet, chaque élément ajouté dans un rapport via IReport va générer automatiquement une balise XML (ou plusieurs) qui définit cet élément.

Nous avons étudié la solution du parseur DOM qui est un outil logiciel permettant de parcourir un document XML et d'en extraire des informations. De plus, il fabrique une représentation en mémoire du document XML.

Document (DOM)

### Architecture

Source XML

TRAITEMENT

PARSEUR

## 

Le fichier source XML va être analysé par le parseur DOM qui va structurer le document en arbre "d'objets".

Cet arbre représente donc l'organisation et le contenu du document XML et permet de faciliter sa manipulation.

# Conclusion

L'objectif de ce rapport était de fournir un document permettant de résumer l'intégration d'un rapport Jasper dans une aplication Java, mais surtout de traiter le problème des requêtes à la volée.

Pour sa rédaction, plusieurs méthodes ont donc été étudiées et testées avant d'être exposées ici.

Il s'avèrerait, à l'heure actuelle, que la méthode des paramètres ne convienne pas, et elle ne sera donc pas utilisée dans un premier temps.

Les deux autres méthodes nous paraissent cependant intéressantes et la combinaison des deux peut également être envisagée.