

## Part I

# Introduction to Reporting with JasperReports

*Estimated Time: 5-10 minutes*

## 1 Introduction

JasperReports is the most popular open-source reporting engine, allowing reports to be generated in a variety of formats including PDF, HTML, Excel, and more. Additionally, it can be embedded within a Java program to display reports right in the GUI of the program. It is a direct competitor to popular, closed-source options like Crystal Reports and other open-source packages such as BIRT.

JasperReports is capable of reading data from a variety of sources, such as databases, XML or CSV files, Excel spreadsheets, and even collections of Java objects.

In this lab, we'll see how to use JasperReports to generate PDF and HTML reports using a list of `Customer` objects stored in an `ArrayList`.

## 2 Getting Started

1. Change to your individual Git repository, create a `lab6` directory, and change to it:

```
$ cd ~/courses/cs2212/labs
$ mkdir lab6
$ cd lab6
```

2. Create a file `pom.xml` with the following elements:

- `groupId`: `ca.uwo.csd.cs2212.USERNAME`
- `artifactId`: `USERNAME-lab6`
- `version`: `1.0-SNAPSHOT`

Refer to lab 2 for the full details of creating a `pom.xml` (remember that you will also need a `modelVersion` tag). Of course, `USERNAME` should be replaced with your UWO username in **lower case**.

3. Edit the `pom.xml` file so that the JAR file that it produces is both executable and *fat* (refer to lab 2). Set the main class to `ca.uwo.csd.cs2212.USERNAME.App`.
4. Add the following dependencies to your `pom.xml` (inside the `project` tag):

```
<dependencies>
  <dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>5.5.0</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
```

```
<artifactId>javax.servlet-api</artifactId>
<version>3.0.1</version>
</dependency>
</dependencies>
```

5. Create the directory structure to host our project:

```
$ mkdir -p src/{main,test}/{java,resources}
```

6. Create the directory structure to house your Java package:

```
$ mkdir -p src/main/java/ca/uwo/csd/cs2212/USERNAME
```

Again, `USERNAME` should be your UWO username in **lowercase**.

7. Download `Customer.java` and `CustomerProvider.java` from the following Gist and put them in the proper directory: <https://gist.github.com/jsuwo/9167132>.

- Customize the package statements in each file.

8. Create a class `App.java` in the proper directory. Add an empty `main` method to it for now. We'll add to it later.

9. Package your code as a JAR. We'll need the JAR file when creating the report in the next section.

## Part II

# Creating a Report Template

*Estimated Time: 60-90 minutes*

In most reporting tools, including JasperReports, reports are generated from *templates*. With JasperReports, we can either create a report template in Java code, using `net.sf.jasperreports.engine.design.JasperDesign` objects, or we can create a special type of XML file (a `jrxml` file) that defines a report template.

Specifying a template in an XML file is more extensible, since we can change the template without having to recompile our code. In this lab, we'll decide to prefer the extensibility offered by XML templates, and focus our attention solely on specifying our report templates in XML.

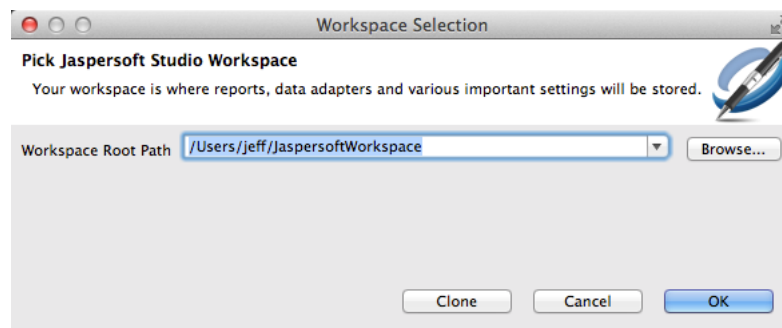
Of course, since XML is simply a text format, we could create our templates with any text editor we like. However, we might like to use some sort of graphical designer that would allow us to drag and drop elements onto our template, and allow us to see the appearance of the template. Fortunately, an open-source, Eclipse-based designer is available for exactly this purpose: Jaspersoft Studio.

In this part, we'll see how to generate a report template using Jaspersoft Studio. We will assume we are building a reporting application for the Accounts Receivable department of ACME Internet, a fictional Internet Service Provider (ISP).

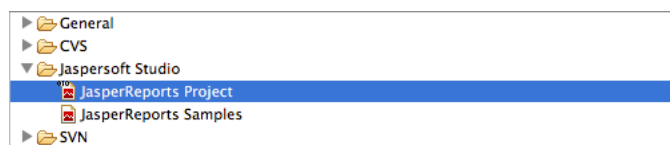
### 3 Creating a Jaspersoft Studio Project

In this section, we'll download and install Jaspersoft Studio, and create a new reporting project within it.

1. Go to <http://community.jaspersoft.com/project/jaspersoft-studio> and download/install the latest version of Jaspersoft Studio (5.5.0, at the time of this writing).
2. Open Jaspersoft Studio and choose a **Workspace Root Path** when prompted. This is where your reporting projects will be stored.

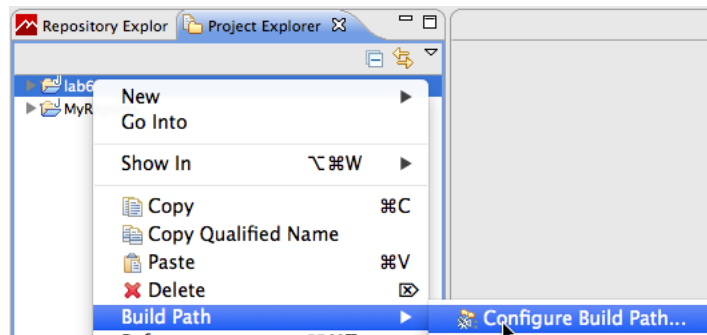


3. Select **File > New > Project**.
4. Select **Jaspersoft Studio > JasperReports Project** and click **Next**.

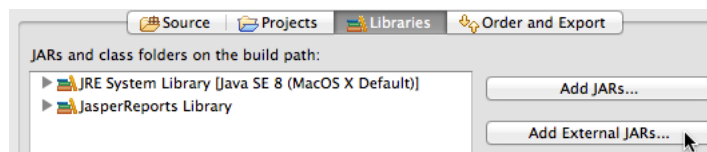


5. Name the project `lab6` and click **Finish**.
6. If the project does not appear, make sure you close the **Welcome** tab.

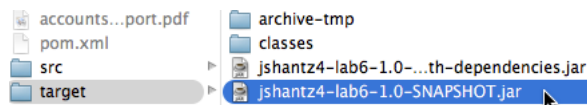
- Right-click on the `lab6` project in the **Project Explorer** and select **Build Path > Configure Build Path...**



- Select the **Libraries** tab and click **Add External JARs...**



- Browse to the `target` directory of the `lab6` directly created in Section 2 and select the `thin` jar (the JAR without dependencies included). Click **Open**.

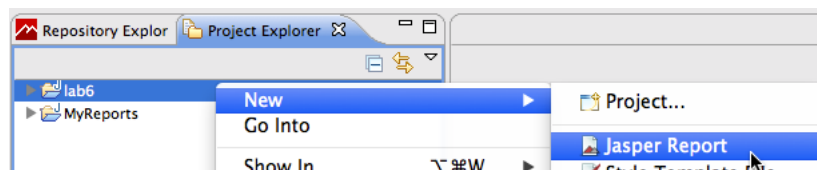


- Click **OK** to close the project properties dialog.

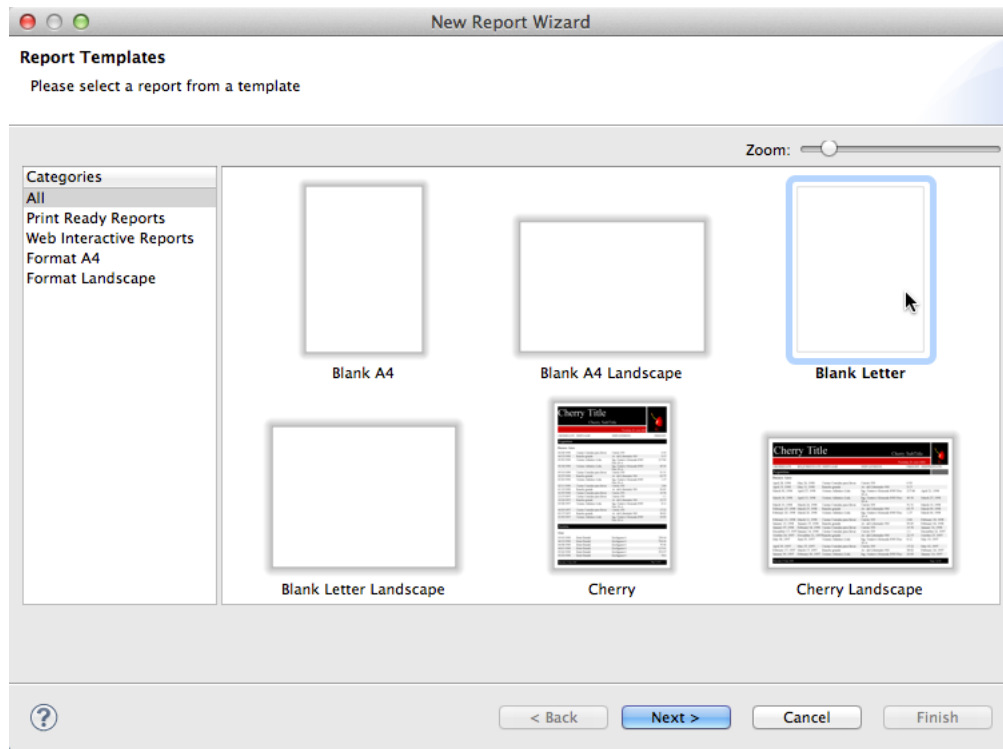
## 4 Adding a Report to the Project

We now have an empty Jaspersoft project and want to begin creating our report. In this section, we'll create a new report to be used for our Accounts Receivable application.

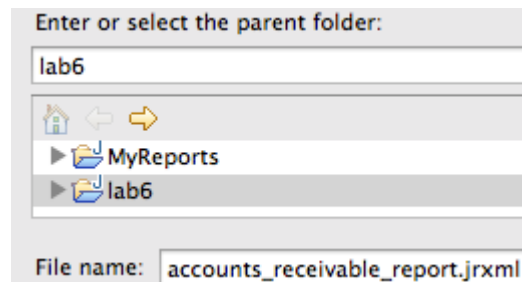
- Right-click on the `lab6` project in the **Project Explorer** and select **New > Jasper Report**.



- Observe that there are a number of pre-defined templates that we can use. To gain practice in report design, we'll stick to using a blank template. Select **Blank Letter** and click **Next >**.



3. Ensure that the `lab6` project is selected, and name the file `accounts_receivable_report.jrxml`. Click **Next**.



We now have to tell Jaspersoft Studio where to retrieve the data that will be put into the report. Once we're finished designing the report, we'll embed the report in a Java application. For now, though, we want to be able to design the report and preview it with real data, so we need to set up a *Data Adapter* so that the IDE knows where to get the report's data.

There are all sorts of data sources that can be used with JasperReports: CSV files, databases, XML documents, etc. We can even have JasperReports pull its data from a collection of Java objects. Because you will be working with object serialization in your project, we will opt for this type of data source.

In Section 2, you created a JAR file containing a class `CustomerProvider`. This class contains a method `loadCustomers` that returns a `Collection` of `Customer` objects:

```

7 public class CustomerProvider {
8
9     public static Collection<Customer> loadCustomers() throws Exception {
10
11         Collection<Customer> customers = new ArrayList<Customer>();
12
13         customers.add(new Customer("Lloyd","Bryant","Male","Plan 1: Lite",0.0));
14         customers.add(new Customer("Sharon","Vincent","Female","Plan 1: Lite",0.0));
15         .
16         .
17         .

```

```

18     return customers;
19 }
20 }

```

The `loadCustomers` method simply creates a new `Collection` of `Customer` objects, adds a number of customers to the collection, and returns the collection.

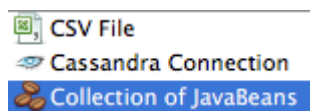
The `Customer` objects themselves store a customer's first and last names, along with the customer's gender, current plan, and balance owing.

JasperReports is able to use the `CustomerProvider` class as a data source for reports. Specifically, any class that contains a `public`, `static` method that returns a `Collection` of objects can be used as a data source.

4. Click **New...** to create a new Data Adapter.

5. Select **Collection of JavaBeans** and click **Next**.

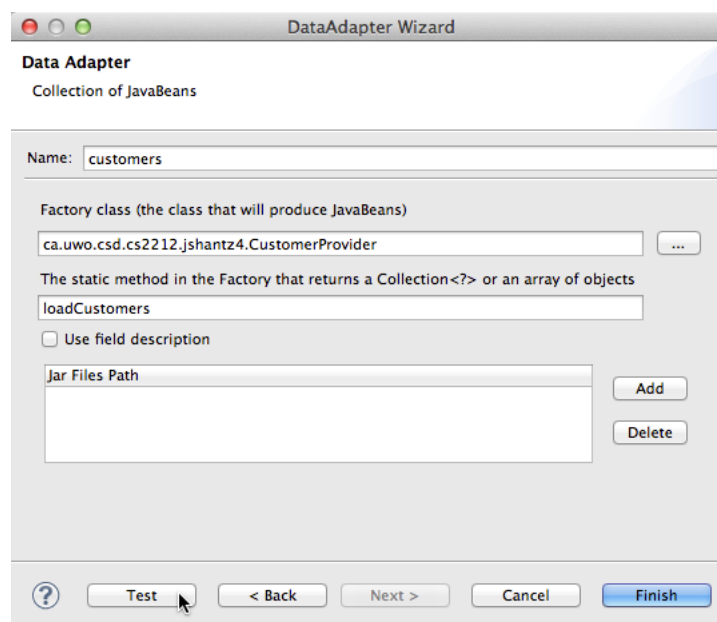
- A *JavaBean* is a class that conforms to the convention of using `getXXX` methods for accessors and `setXXX` methods for mutators. The `Customer` class conforms to this convention.



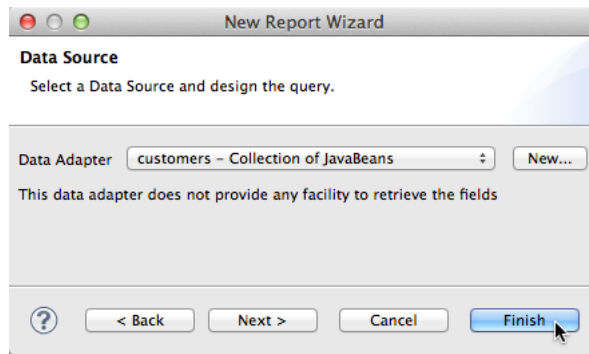
6. Enter the following information:

<b>Name</b>	customers
<b>Factory class</b>	ca.uwo.csd.cs2212.USERNAME.CustomerProvider
<b>Static method</b>	loadCustomers

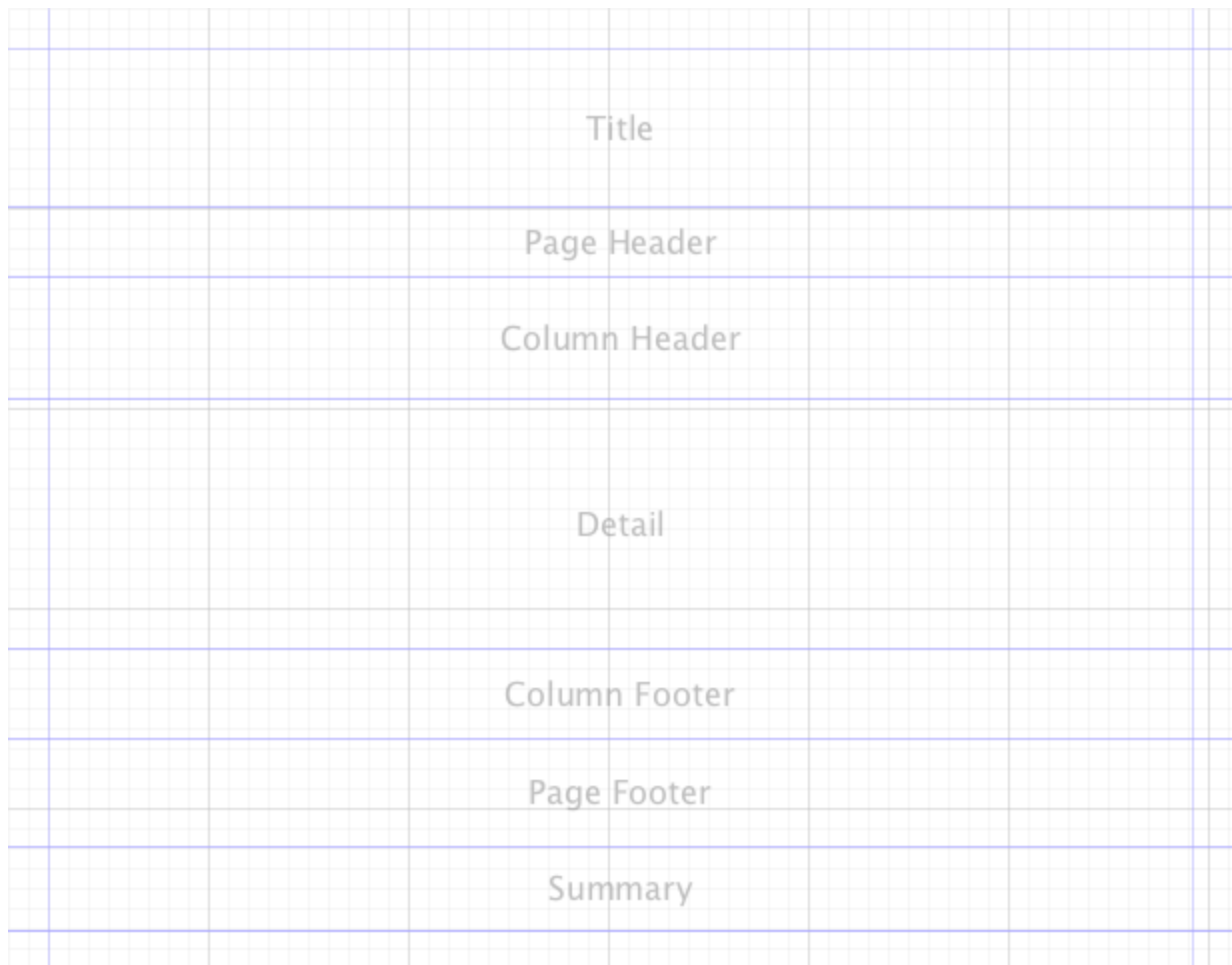
7. Click the **Test** button and ensure that it reports a successful test. If not, check your typing and try again.



8. Click **Finish**. Ensure that `customers` is the selected Data Adapter and click **Finish** again.



We now have an empty report that we can start designing. Observe that there are numerous *bands* displayed in the report:



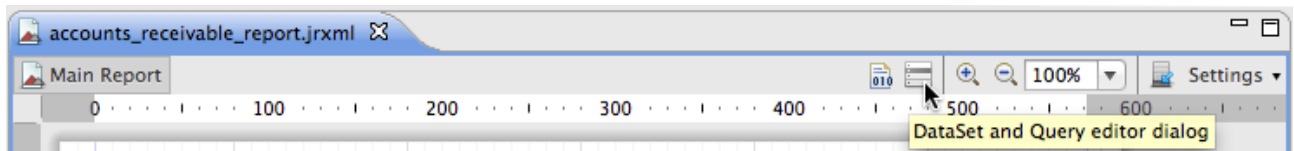
Anything placed in the *Title* band will appear at the top of the report, while elements placed in the *Page Header* band will appear at the top of each page. Similarly, the various other bands control where elements will appear in the report.

In the next section, we'll start designing our report.

## 5 Configuring the Fields of the Report

We have configured a data source for the report, but we need to tell JasperReports which *fields* we wish to use from the data source. If our data source was a database, we would need to specify the database columns to include in our report. Since we're using a collection of objects, our fields correspond to the instance variables in the `Customer` class. In this section, we'll configure the fields of our report.

1. Open the **DataSet and Query Editor** dialog.



2. Select the **Java Bean** tab.
3. In the **Class Name** field, enter the fully-qualified name of the **Customer** class.

Query Java Bean

Class Name

balanceOwing  
class  
firstName  
gender  
lastName  
planName

4. Select all fields except **class** and click **Get Selected Fields**. The fields should appear in the table at the bottom of the dialog.

Field Name	Class Type	Description
balanceOwing	java.lang.Double	balanceOwing
firstName	java.lang.String	firstName
gender	java.lang.String	gender
lastName	java.lang.String	lastName
planName	java.lang.String	planName

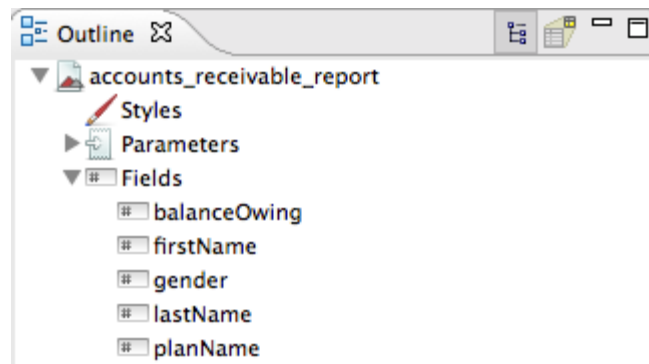
Fields Parameters Sorting Filter Expression Data preview

5. Select the **Data preview** tab and click **Refresh Preview Data**. After a moment, you should see the data from the collection appear.

balanceOwing	firstName	gender	lastName
0.0	Lloyd	Male	Bryant
0.0	Sharon	Female	Vincent
0.0	Lyle	Male	Strannemar
190.06	Omar	Male	Akai
66.9	Alissa	Female	Telos
11.06	Jamie	Male	Welker
0.0	Hinda	Female	Whiteman
226.99	Allain	Male	Fleury
70.28	Rebekkah	Female	Bowab

6. Click **OK**. You should now see the fields appear in the **Outline** pane.

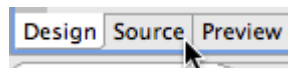




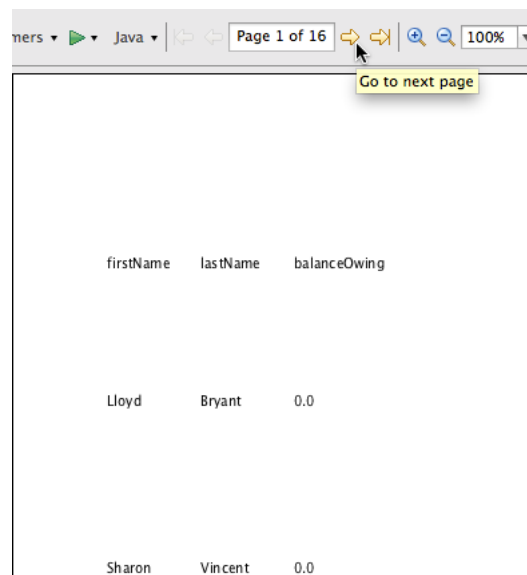
7. Drag the `firstName`, `lastName`, and `balanceOwing` fields from the **Outline** to the *Detail* band to create three *Text Elements*.

firstName	lastName	balanceOwing
Column Header		
\$F{firstName}	\$F{lastName}	\$F{balanceOwin
Detail		

8. Select the **Source** tab at the bottom of the pane. Observe that the report is merely an XML file. Take a moment to explore its contents, but do not change them.



9. Select the **Preview** tab at the bottom of the pane. You can use the navigation controls at the top of the pane to navigate through the pages of the report.

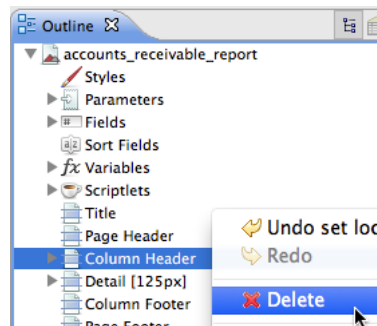


Our report doesn't look like much yet, but we've hooked it up to our data source and demonstrated that we can display data from a collection of Java objects. In the next section, we'll spend a few moments making things look a little nicer.

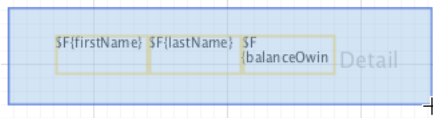
## 6 Styling the Report

We created a bare-bones report in the last section, but it leaves much to be desired. First, there is a considerable amount of space between each customer record. Second, the `balanceOwin` field should be formatted as currency and, since this is a report for an Accounts Receivable department, we might want to style positive account balances in red. Finally, we likely don't need the column headers, so we might want to remove them. In this section, we'll deal with all of these issues.

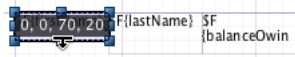
1. Select the **Design** tab at the bottom of the pane.
2. In the **Outline** pane, right-click the **Column Header** band and select **Delete**.
  - The **Column Header** band will disappear from the report. Feel free to select the **Preview** tab at any time to see how a change affects the appearance of the report.



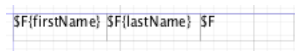
3. Repeat the process for the **Column Footer** band.
4. Click any empty space in the **Detail** band and drag to select all three fields.



5. Move the fields to the top-left corner of the **Detail** band.
6. Select the `firstName` field and reduce its height to 20 pixels.



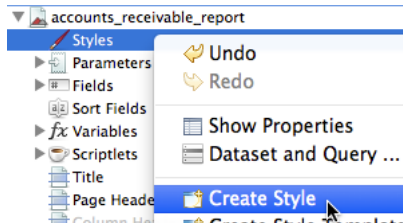
7. Select all three fields again, and right-click on any one of them. Select **Size Components > Match Height (Min)**.
  - All three fields should now be the same height.



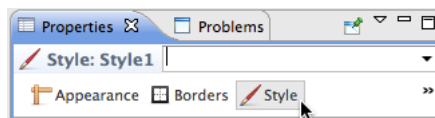
8. Preview the report. Notice that the records are still spaced quite far apart.
9. Switch back to the **Design** tab. Drag the blue line at the bottom of the **Detail** pane up until it is 20 pixels.



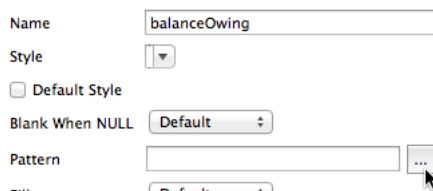
10. Preview the report again. Notice now that the records are more reasonably spaced.
11. In the **Outline** pane, right-click **Styles** and click **Create Style**.



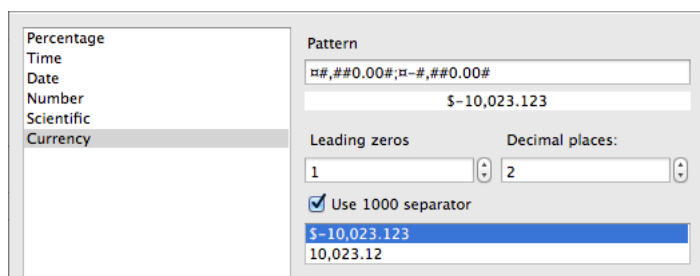
12. Select the newly-created style in the **Outline** pane ( **Style1** ) and click the **Style** button in the **Properties** pane.



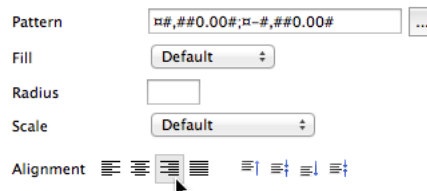
13. Name the style **balanceOwing** and click the ... button beside the **Pattern** field.



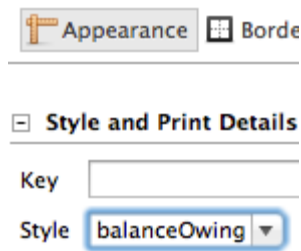
14. Select **Currency** and select the first pattern in the list.
15. Change the **Decimal places** to **2** and click **Finish**.



16. Change the **Alignment** for the style to **Right**.



17. Select the **balanceOwing** field in the **Detail** band.
18. In the **Properties** pane, scroll down on the **Appearance** tab and set the **Style** to the new style just created.

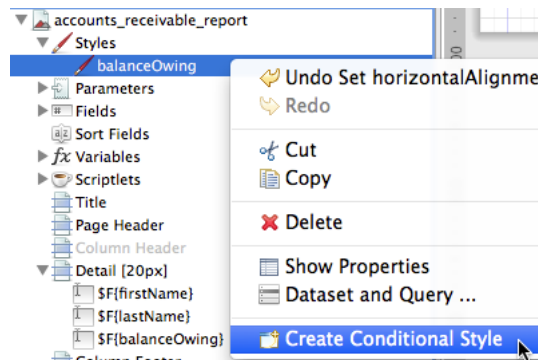


19. Preview the report. Values in the **balanceOwing** column should now appear with a currency symbol and two decimal places, and they should be right-aligned.

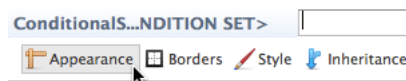
Note that it wasn't necessary to create a style to accomplish what we just did. We could have set the pattern and alignment of the **balanceOwing** field directly. However, doing so would only affect that field. We might want to use this particular style on currency fields later. By creating a style, we can apply it to as many fields as we like.

The last thing we need to do with this style is to create a *conditional style*. This will allow us to style values in the column in red text when they are greater than zero (i.e. when the customer owes money to the company).

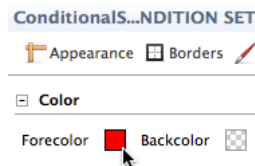
20. Right-click the **balanceOwing** style in the **Outline** pane and select **Create Conditional Style**.



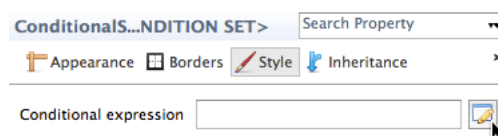
21. Select the newly-created condition in the **Outline** pane and select the **Appearance** tab in the **Properties** pane.



22. Set the foreground colour to red.

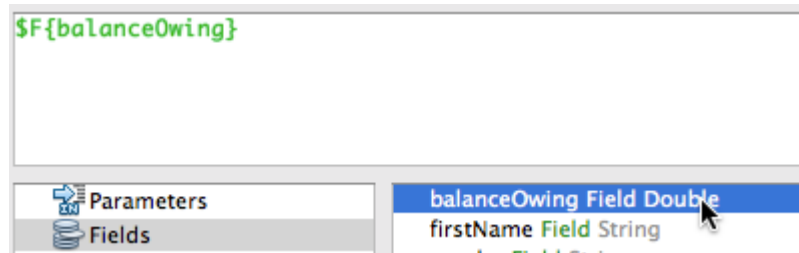


23. Click the **Style** tab and click the button beside the **Conditional expression** field.



24. Select **Fields** and double-click the `balanceOwing` field.

- `$F{balanceOwing}` should be added to the expression editor.
- By enclosing a field name in `$F{}`, we can reference its value in an expression



25. Add `> 0` to the end of the expression and click **Finish**.



We have now added a condition to the `balanceOwing` style such that, for any element to which we apply the `balanceOwing` style, its text will be coloured red if the value of `balanceOwing` is positive.

26. Preview the report and confirm that positive balances appear in red, while zero balances appear as usual in black.

Lloyd	Bryant	\$0.00
Sharon	Vincent	\$0.00
Lyle	Strannemar	\$0.00
Omar	Akai	\$190.06
Alissa	Telos	\$66.90
Jamie	Welker	\$11.06
Hinda	Whiteman	\$0.00

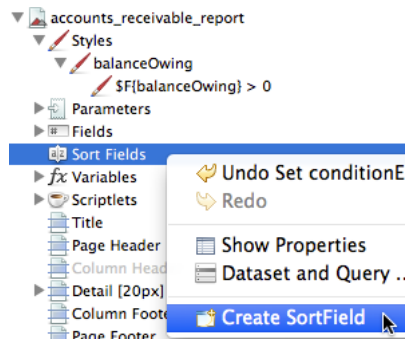
## 7 Sorting and Grouping

In the last section, we improved the appearance of our report. Still, the records are presented in the order in which they appear in the data source, which isn't very intuitive. We might want to sort by each customer's balance owing.

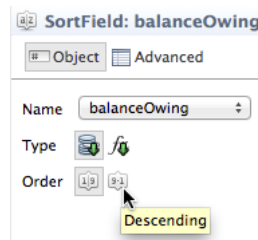
Additionally, it might be more informative to group customers according to their plan. This way, we can view the balances owing by plan, which might tell us which customer groups tend to pay their bills on time. This might influence certain business decisions, such as offering promotions to certain groups of customers, or more aggressively targeting other groups of customers for collections.

In this section, we'll tackle both of these issues as we explore sorting and grouping.

1. In the **Outline** pane, right-click on **Sort Fields** and select **Create SortField**.



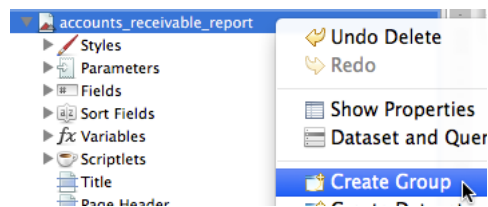
2. Select `balanceOwing` and click **Finish**.
3. Create another sort field for `lastName`.
4. Select the `balanceOwing` sort field in the **Outline** pane and change its order to **Descending** in the **Properties** pane.



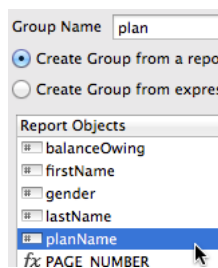
5. Preview the report. The records should now be presented in descending order by balance owing, and then in ascending order by last name.

Mathilde	Siegel	\$246.29
Terri	MacEvoy	\$235.82
Zachary	Oastler	\$228.11
Allain	Fleury	\$226.99
Randal	Bolito	\$224.03
Lynn	McCartney	\$222.35
Hal	Beehler	\$213.42
Omar	Akai	\$190.06

6. Right-click on `accounts_receivable_report` in the **Outline** pane and select **Create Group**.



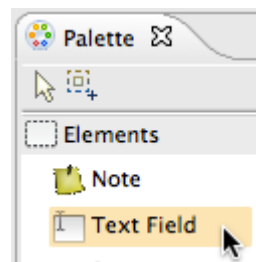
7. Name the group `plan` and select the `planName` field.



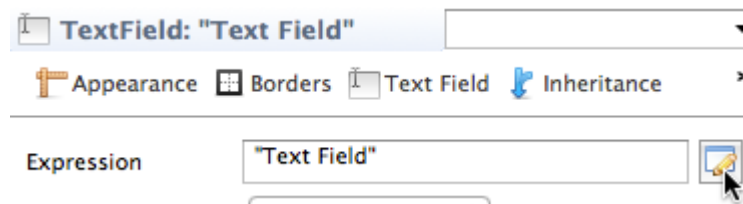
8. Click **Finish**. Notice that two new bands have been added to the report: **plan Group Header** and **plan Group Footer**.

plan Group Header			
\$F{firstName}	\$F{lastName}	\$F	Detail
plan Group Footer			

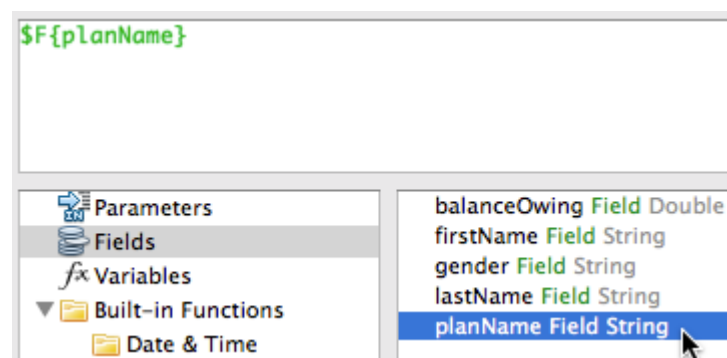
9. From the **Palette** pane, drag a **Text Field** element to the **plan Group Header** band.



10. Select the new text field and click the **Text Field** tab in the **Properties** pane. Click the button beside the **Expression** field.



11. Delete the text from the expression editor. Select **Fields** and double-click **planName**. Click **Finish**.



12. Preview the report. Notice that the plan name now appears above each list.

Plan 2:  
Express

Mathilde	Siegel	\$246.29
Terri	MacEvoy	\$235.82
Zachary	Oastler	\$228.11

Plan 1: Lite

Allain	Fleury	\$226.99
--------	--------	----------

Plan 2:  
Express

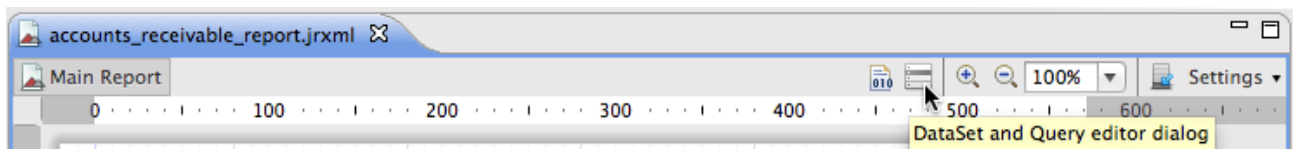
Randal	Bolito	\$224.03
Lynn	McCartney	\$222.35
Hal	Beehler	\$213.42

We have a few problems here. First, the plans are out of order. Notice that we have a number of *Plan 2* customers, followed by a *Plan 1* customer, and then more *Plan 2* customers. This is because we need to set the report to sort first on `planName`, followed by the rest of our existing sort fields.

Second, the group header needs to be widened so that it doesn't span multiple lines, and perhaps needs some formatting applied to it.

We'll address these problems next.

13. Open the **DataSet and Query Editor** dialog.



14. Select the **Sorting** tab and click **Add**.
15. Select `planName` and click **Finish**. This is another way to add a sort field.
16. Select the `planName` sort field and click the **Up** button until it is at the top.
  - This way, the records will be sorted first by plan name, then balance owing, and, finally, last name.

Field Name	Type	Order	
<input checked="" type="checkbox"/> planName	Field	Ascending	<input type="button" value="Add"/>
<input checked="" type="checkbox"/> balanceOwing	Field	Descending	<input type="button" value="Delete"/>
<input checked="" type="checkbox"/> lastName	Field	Ascending	<input type="button" value="Up"/>
			<input type="button" value="Down"/>



17. Click **OK** and preview the report. The customers should now be grouped properly by plan.

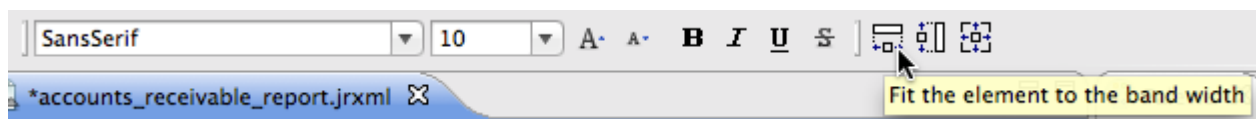
Plan 1: Lite

Allain	Fleury	\$226.99
Omar	Akai	\$190.06
Rebekkah	Bowab	\$70.28
Alissa	Telos	\$66.90
Jamie	Welker	\$11.06
Lloyd	Bryant	\$0.00
Lyle	Strannemar	\$0.00
Sharon	Vincent	\$0.00
Hinda	Whiteman	\$0.00

Plan 2:  
Express

Mathilde	Siegel	\$246.29
Terri	MacEvoy	\$235.82
Zachary	Oastler	\$228.11
Randal	Bolito	\$224.03
Lynn	McCartney	\$222.35

18. Select the **planName** element in the **plan Group Header** band and click the toolbar button to fit the element to the band width.



19. Adjust the font size of the element (perhaps to 18) and bold its text.

<b>\$F{planName}</b>			plan Group Header
\$F{firstName}	\$F{lastName}	\$F	Detail

20. Resize the **plan Group Header** band so that there is not so much space between it and the **Detail** band.

<b>\$F{planName}</b>			plan Group Header
\$F{firstName}	\$F{lastName}	\$F	Detail

21. Preview the report. The group headers should now look a little more palatable.

## Plan 1: Lite

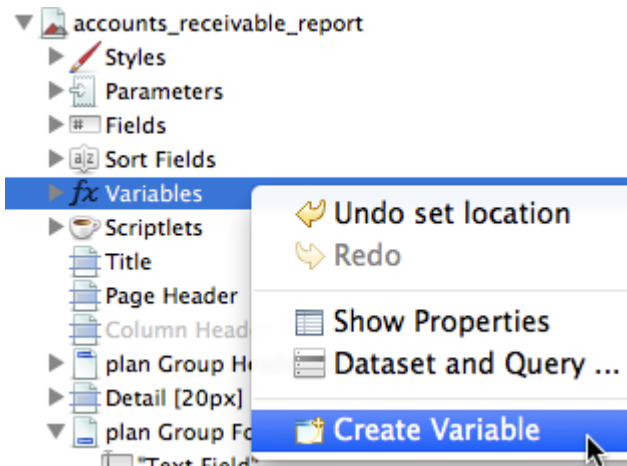
Allain	Fleury	\$226.99
Omar	Akai	\$190.06
Rebekkah	Bowab	\$70.28
Alissa	Telos	\$66.90
Jamie	Welker	\$11.06
Lloyd	Bryant	\$0.00
Lyle	Strannemar	\$0.00
Sharon	Vincent	\$0.00
Hinda	Whiteman	\$0.00

## Plan 2: Express

Mathilde	Siegel	\$246.29
Terri	MacEvoy	\$235.82
Zachary	Oastler	\$228.11
Randal	Bolito	\$224.03
Lynn	McCartney	\$222.35
Hal	Beehler	\$213.42

Lastly, we might want to add a group footer element to sum the balances owing for each plan. For this, we will need to create a *variable* that maintains the sum for each group.

22. Right-click on **Variables** in the **Outline** pane, and select **Create Variable**.



23. Select the newly-created variable in the **Outline** and, in the **Properties** pane, enter the following information:

Name	groupBalance
Value Class Name	java.lang.Double
Calculation	Sum
Expression	\$F{balanceOwing}
Reset type	[Group] plan

**fx Variable: groupBalance** Search Property ▾


☒ Object ☐ Advanced


---

Name

Value Class Name  ...

Calculation  ▾

Expression  

Initial Value Expression  

Increment type  ▾

Incrementer Factory Class Name  ...

Reset type  ▾




Here, we've created a variable that will sum the `balanceOwing` field for each customer, and will reset back to zero at the start of each group. Note that we could have used the expression editor to specify the expression of the variable (as we did several times earlier), but we can directly type in the expression, as we've done here.

24. Drag the `groupBalance` variable from the **Outline** to the **plan Group Footer** band. Line it up with the `balanceOwing` field in the **Detail** band, but leave some room between it and the top of the band.

<b>\$F{planName}</b>		
\$F{firstName}	\$F{lastName}	\$F{...}
		\$V{groupBalance}

25. Select the new element and apply the `balanceOwing` style to it in the **Properties** pane.

**TextField: \$V{groupBalance}**

 Appearance  Borders  Text

---

Forecolor ☒ Backcolor ☐

☒ Transparent

☐ **Style and Print Details**

---

Key

Style  ▾

26. Bold the text in the new element by either clicking the button in the toolbar, or selecting the **Text Field** tab in the **Properties** pane and clicking the appropriate button.
- Observe that, even though we applied a style to the element, we can also apply further formatting to the element directly.

27. Preview the report and observe that we now have a styled column footer for the `balanceOwing` column, which sums the balances of customers in each plan.

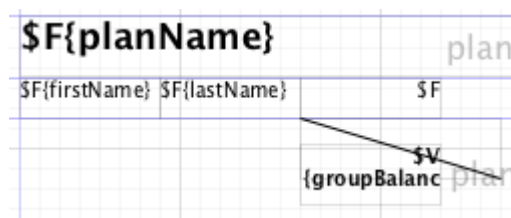
### Plan 1: Lite

Allain	Fleury	\$226.99
Omar	Akai	\$190.06
Rebekkah	Bowab	\$70.28
Alissa	Telos	\$66.90
Jamie	Welker	\$11.06
Lloyd	Bryant	\$0.00
Lyle	Strannemar	\$0.00
Sharon	Vincent	\$0.00
Hinda	Whiteman	\$0.00
		\$565.29

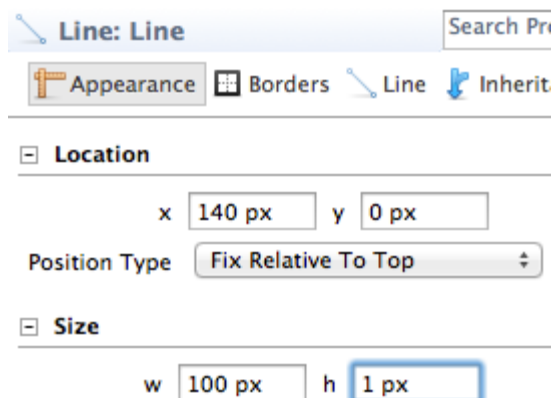
### Plan 2: Express

Mathilde	Siegel	\$246.29
Terri	MacEvoy	\$235.82
Zachary	Oastler	\$228.11
Randal	Bolito	\$224.03
Lynn	McCartney	\$222.35
Hal	Beehler	\$213.42

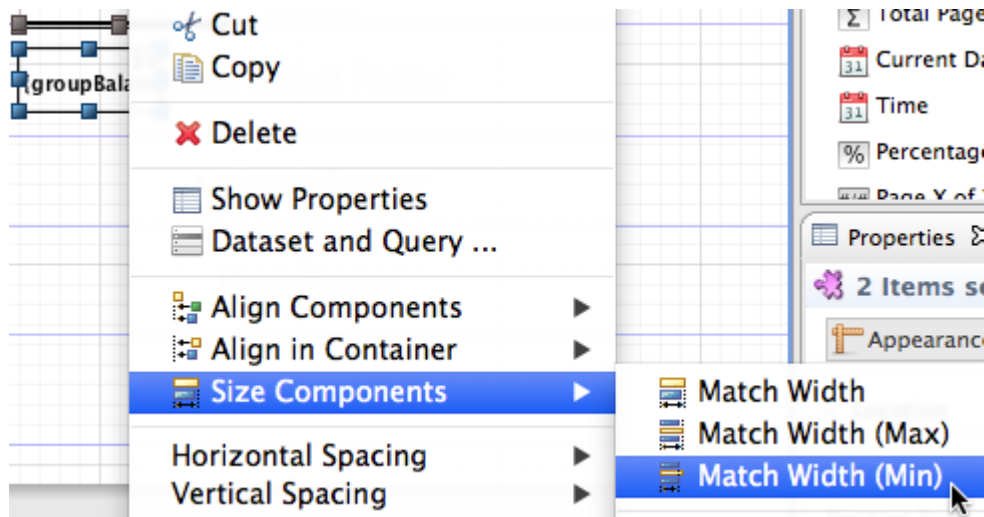
28. Drag a **Line** element from the **Palette** right above the `groupBalance` element in the **plan Group Footer**.



29. Select the line and change its height to `1px` in the **Properties** pane.



30. Select both the `groupBalance` field and the line. Right-click on one of the elements and select **Size Components** > **Match Width (Min)**.



31. Preview the report. You should now have a horizontal line separating the `balanceOwing` column from its sum in the group footer.

### Plan 1: Lite

Allain	Fleury	\$226.99
Omar	Akai	\$190.06
Rebekkah	Bowab	\$70.28
Alissa	Telos	\$66.90
Jamie	Welker	\$11.06
Lloyd	Bryant	\$0.00
Lyle	Strannemar	\$0.00
Sharon	Vincent	\$0.00
Hinda	Whiteman	\$0.00
		<hr/>
		\$565.29

Our report is starting to take shape. In the next section, we'll add a header to the report.

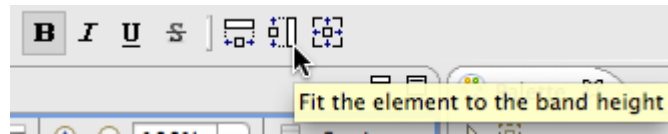
## 8 Adding a Report Header

In this section, we'll continue to improve the appearance of our report by adding a header that will be displayed at the top of the first page of the report.

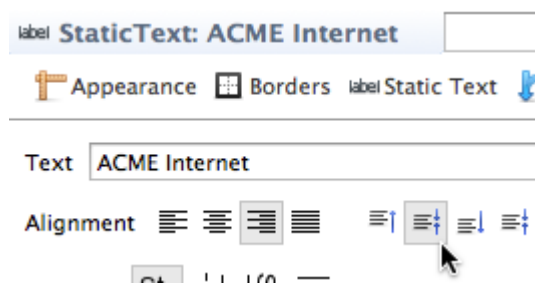
1. Drag a **Static Text** element from the **Palette** to the **Title** band.
2. Double-click the element and change its text to `ACME Internet`.
3. Change its font size to `24` and bold its text.
4. Resize the width of the element accordingly.



5. Select the element and click the toolbar button to fit the element to the height of the band.

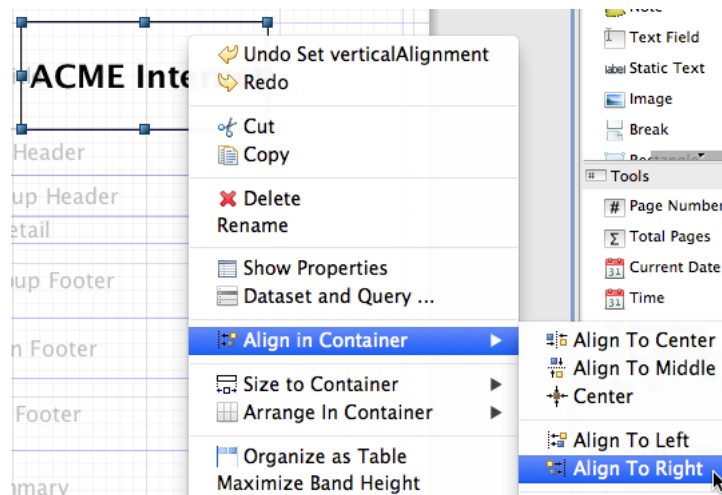


6. In the **Properties** pane, select the **Static Text** tab and change the horizontal alignment to **Right** and the vertical alignment to **Middle**.

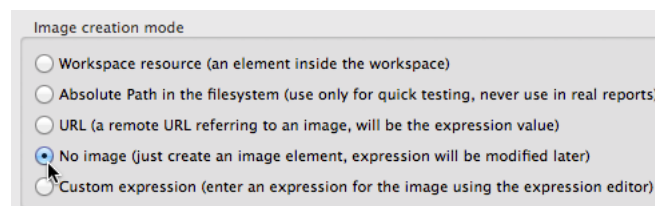


7. Right-click the text element and select **Align in Container > Align to Right**.

- The text field should now be aligned to the right side of the report.



8. Drag an **Image** element from the **Palette** to the **Title** band.
9. Select **No image** and click **OK**.

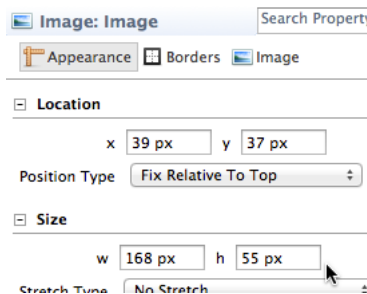


You might be wondering why we selected **No image**. We want to add a logo to the report, but we don't want to hard code a path to an image on the current system. After all, this report will be bundled into a JAR file and the JAR might be installed on numerous systems. Ideally, we'd like to load the logo at run-time from the JAR file.

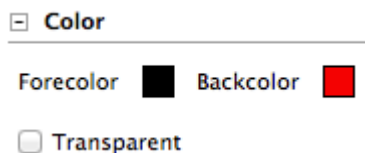
To do this, we have set the **Image creation mode** to **No image** and we will create a *parameter*, which will allow us to pass in an **Image** object from our Java program to the report at run-time. This way, our Java program can load the image from the JAR file and then pass it to the report to be rendered.

We'll create a parameter in a few moments, but let's finish styling our image.

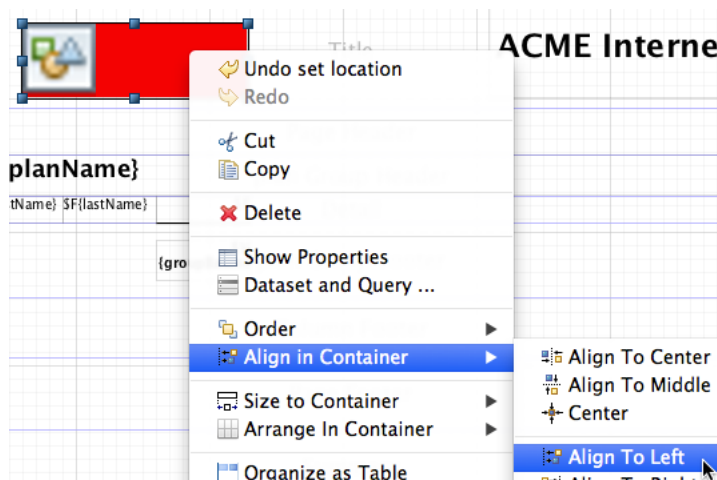
10. Select the image and change its size to **168px** by **55px** in the **Properties** pane.



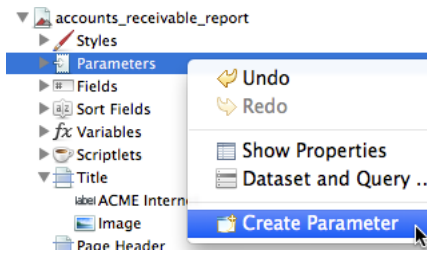
11. Change its **Backcolor** to red, and uncheck the **Transparent** checkbox. This will make the image element red, allowing us to see it when we preview the report.



12. Right-click on the image and select **Align in Container > Align to Left**.



13. Repeat this process, this time selecting **Align to Middle**, to vertically center the image in the **Title** band.
14. Right-click **Parameters** in the **Outline** pane and select **Create Parameter**.

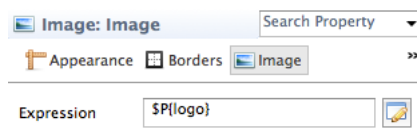


15. Select the parameter. In the **Properties** pane, enter the following information:

<b>Name</b>	logo
<b>Class</b>	java.lang.Object
<b>Is For Prompting</b>	false (unchecked)

16. Select the image element. In the **Properties** pane, select the **Image** tab and enter `$P{logo}` in the **Expression** field.

- This links the image to the `logo` parameter. Whatever image we pass in as the `logo` parameter will be displayed.



17. Add another **Static Text** element below the logo. Set its properties as follows:

<b>Text</b>	Accounts Receivable Report
<b>Font Size</b>	20
<b>Font Weight</b>	Bold
<b>Horizontal Alignment</b>	Center
<b>Vertical Alignment</b>	Middle

18. Click the toolbar button to stretch the width of the element to the width of the **Title** band. Note that you may have to increase the height of the **Title** band and increase the height of the text element in order to see its text.

19. Preview the report. You should have a completed header.



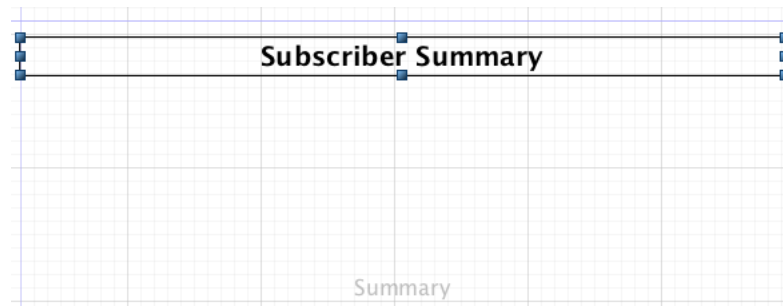
Now that we have a report header, we'll finish up in the next few sections by creating a footer.

## 9 Summarizing Data

In this section, we'll add some summary data to the **Summary** band that will sum the balances owing of all customers. We will also provide counts of the total number of customers, as well as the number of male and female customers – demographic information that might be of interest for marketing purposes.



1. Resize the **Summary** band so that we have a good deal of space to work with – perhaps about 400 px (drag down the blue line at the bottom).
2. Add a horizontally and vertically centered **Static Text** element to the top of the **Summary** band, with the text **Subscriber Summary**. Make it bold, with a font size of 20.



3. Create a new variable (right-click **Variables** in the **Outline** pane) with the following properties:

<b>Name</b>	totalBalance
<b>Value Class Name</b>	java.lang.Double
<b>Calculation</b>	Sum
<b>Expression</b>	\$F{balanceOwing}
<b>Reset type</b>	Report

The difference between this variable and the `groupBalance` variable is the *reset type*. For the latter, we specified that the variable should reset after each group. Here, we're specifying that the variable should reset to zero after the report is finished. In other words, the variable should sum every `balanceOwing` value in the entire report.

4. Create three new variables as follows:

<b>Name</b>	subscriberCount	maleCount	femaleCount
<b>Value Class Name</b>	java.lang.Integer	java.lang.Integer	java.lang.Integer
<b>Calculation</b>	Count	Sum	Sum
<b>Expression</b>	\$F{firstName}	\$F{gender} == "Male" ? 1 : 0	\$F{gender} == "Female" ? 1 : 0
<b>Reset type</b>	Report	Report	Report

Notice for the expressions of `maleCount` and `femaleCount`, we are using the *ternary operator*. The `maleCount` expression, for example, says that, for each customer, if the customer's gender is `Male`, then the expression should evaluate to `1`; otherwise, it should evaluate to `0`. We then sum all these values to obtain the total number of male customers. This is similar to the `SUMIF` function in Excel.

For the `subscriberCount` variable, we are just counting all `firstName` values in the report to give us our subscriber count. We could have used any field we wanted here.

5. Drag the four newly-created variables to the **Summary** band. Add four **Static Text** elements and try to make your summary look like the image below.

Subscriber Summary	
Total Balances:	<code>\$V{totalBalance}</code>
Subscriber Count:	<code>\$V{subscriberCount}</code>
Male Subscribers:	<code>\$V{maleCount}</code>
Female Subscribers:	<code>\$V{femaleCount}</code>

You will likely find it useful to select multiple elements, right-click them, and use the **Align Components** and **Size Components** menu items.

6. Apply the `balanceOwing` style to the `totalBalance` element in the **Properties** pane, but change the horizontal alignment of the element to **Left** (thus overriding the right-alignment of the applied style).
7. Preview the report and ensure that your summary data is correct.

Subscriber Summary	
Total Balances:	\$3,483.14
Subscriber Count:	62
Male Subscribers:	30
Female Subscribers:	32

## 10 Adding a Pie Chart

In this section, we'll add a pie chart to the **Summary** band, which will show us the percentage of customers that subscribe to each plan.

To accomplish this, we'll need to create four more variables, which count the number of customers in each of our four plans.

1. Create the following variables:

Name	<code>liteCount</code>
Value Class Name	<code>java.lang.Integer</code>
Calculation	Sum
Expression	<code>\$F{planName} == "Plan 1: Lite" ? 1 : 0</code>
Reset type	Report

---

Name	<code>expressCount</code>
Value Class Name	<code>java.lang.Integer</code>
Calculation	Sum
Expression	<code>\$F{planName} == "Plan 2: Express" ? 1 : 0</code>
Reset type	Report

---

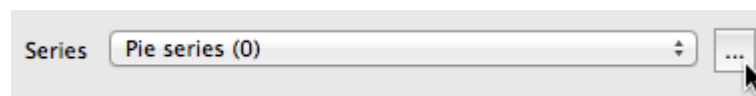
Name	<code>extremeCount</code>
Value Class Name	<code>java.lang.Integer</code>
Calculation	Sum
Expression	<code>\$F{planName} == "Plan 3: Extreme" ? 1 : 0</code>
Reset type	Report

<b>Name</b>	extremePlusCount
<b>Value Class Name</b>	java.lang.Integer
<b>Calculation</b>	Sum
<b>Expression</b>	<code>\$F{planName} == "Plan 4: Extreme Plus" ? 1 : 0</code>
<b>Reset type</b>	Report

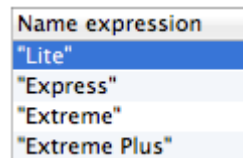
2. Drag a **Chart** element from the **Palette** to the **Summary** band.
3. Select **Pie3D Chart** and click **Next**.



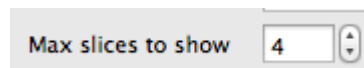
4. Edit the series list by clicking the ... icon.



5. Rename **SERIES 1** to **Lite**.
6. Add three new series: **Express**, **Extreme**, **Extreme Plus** (be sure to put them in double quotes in the expression editor).

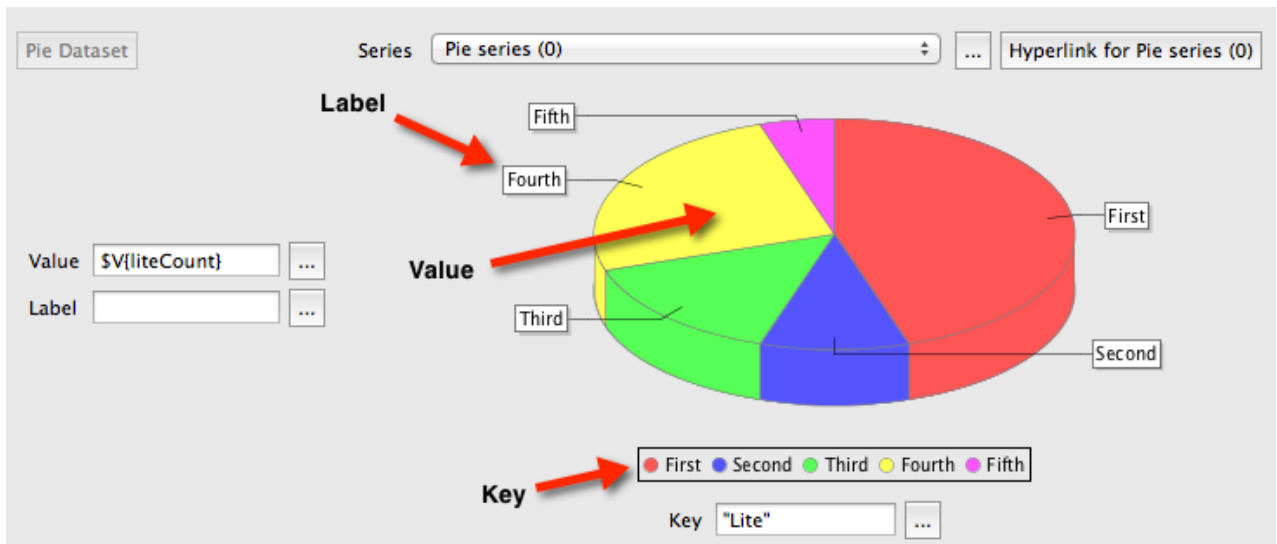


7. Click **OK**. Set the **Max slices to show** to **4** so that all 4 series will be displayed in the pie chart.



8. Ensure that **Pie series (0)** is selected in the **Series** drop down and enter the values below. For now, leave the **Label** blank.

<b>Key</b>	"Lite"
<b>Value</b>	<code>\$V{liteCount}</code>



9. Select **Pie series (1)** from the **Series** drop down and enter the same information, substituting "Express" for "Lite" and `expressCount` for `liteCount`.
10. Repeat the process for series 2 and 3, using "Extreme" and `extremeCount`, and "Extreme Plus" and `extremePlusCount`, respectively.
11. Click **Finish**. Select the chart and click the **Chart Plot** tab in the **Properties** pane. Enter the following:

Label Format	{0} ({2})
Legend Label Format	{0}

**Chart: Pie 3D**

Appearance Borders Hyperlink Chart **Chart Plot** »

Backcolor ☐

Foreground Alpha

Background Alpha

Series Colors  ...

Orientation

Show Labels

Circular

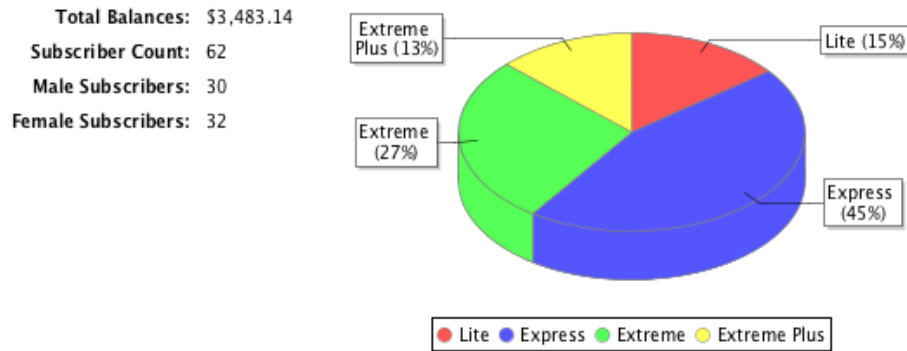
Label Format

Legend Label Format

These values warrant an explanation. JasperReports provides some built-in variables that can be used within the label of a pie chart. `{0}` will display the key of the slice, and therefore will display the plan name. `{1}` will display the value of the slice. Thus, if we were to use it, it would display the count of all customers on each plan. `{2}` will display the percentage that the slice's value represents in the entire pie chart, thus giving us the percentage of our customers that are on each plan.

12. Preview the report. You should now have a pie chart showing the breakdown of our customers by plan.

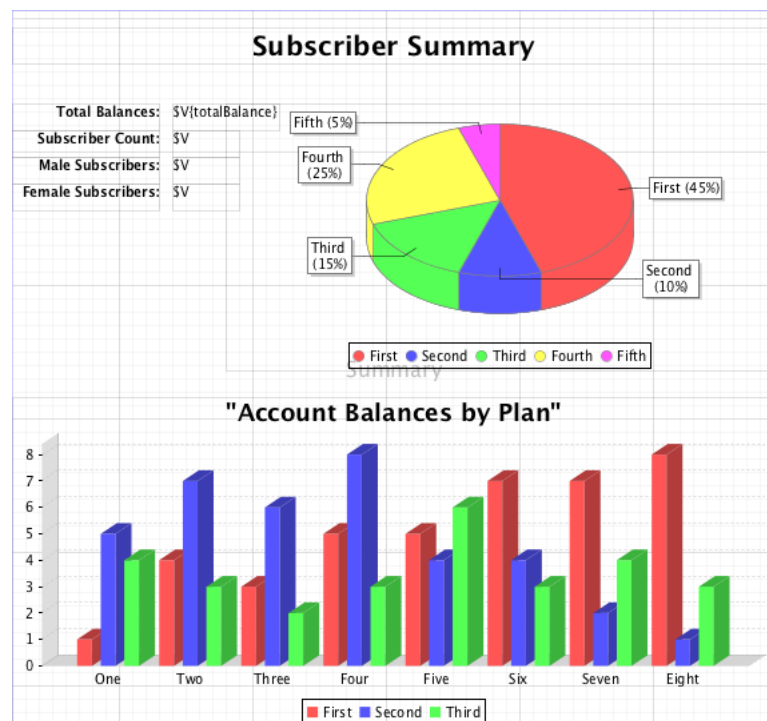
## Subscriber Summary



## 11 Adding a Bar Chart

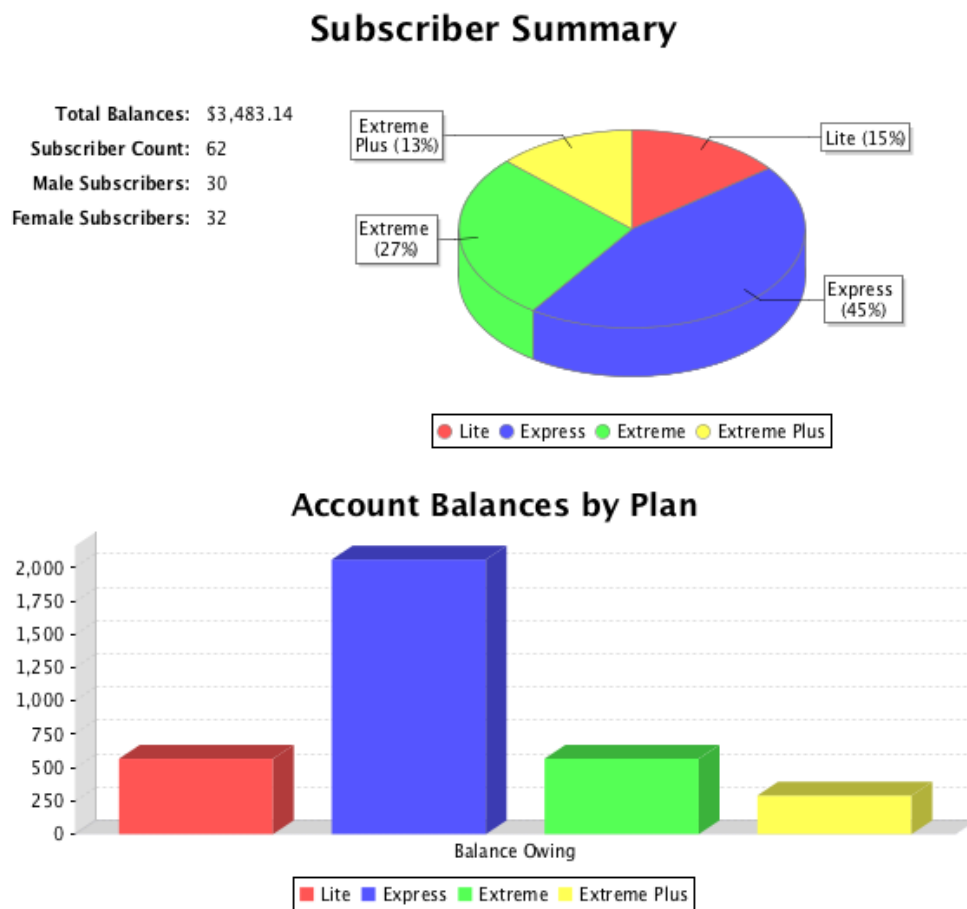
We're nearly finished our report for the Accounts Receivable department. Before we're done, though, our manager would like us to add a bar chart showing the total account balances owing by plan.

1. Drag another **Chart** element to the **Summary** band, below the pie chart. Select **Bar3D Chart** and click **Finish**.
2. Resize the chart to fit the entire width of the page.
3. Add a title of "Account Balances by Plan" (quotes are needed) to the chart by using the **Properties** pane (hint: see the **Chart** tab).



4. Create four new variables: `liteBalance`, `expressBalance`, `extremeBalance`, and `extremePlusBalance`.
  - Each should have a class of `java.lang.Double`

- Each should sum up the `balanceOwing` variable for its respective plan.
    - Hint: use the ternary operator as we saw earlier when creating the variables for the pie chart.
    - Instead of yielding `1` or `0`, yield the value of the `balanceOwing` field or `0`.
5. Double-click the bar chart to open the **Chart Wizard** dialog.
  6. Create 4 series: `Lite`, `Express`, `Extreme`, and `Extreme Plus`.
  7. For each series, set the **Value** field to the associated variable (e.g. `liteBalance` for `Lite`).
    - Don't forget to use the proper syntax to reference the variable. If unsure, use the expression editor.
  8. For each series, set the **Category** to `"Balance Owing"` (quotes are needed).
  9. Click **Finish** and preview your report. You should now have a bar chart summarizing the total balances owing by plan.



## Part III

# Generating a Report from a Java Application

*Estimated Time: 10-15 minutes*

We now have a report designed and it's time to hook it into our Java application so that it can generate and output a report without having to go through Jaspersoft Studio.

## 12 Outputting PDF and HTML Reports

We'll begin by having the program output a PDF report.

1. Copy `accounts_receivable_report.jrxml` from your Jaspersoft Studio project to the `src/main/resources` directory of your `lab6` directory.
2. Edit the `App.java` class and add the following code:

```
1 package ca.uwo.csd.cs2212.USERNAME;
2
3 import java.io.*;
4 import java.util.*;
5 import net.sf.jasperreports.engine.*;
6 import net.sf.jasperreports.engine.data.*;
7 import net.sf.jasperreports.engine.design.*;
8 import net.sf.jasperreports.engine.xml.*;
9
10 public class App {
11
12     private final static String REPORT_FILENAME = "accounts_receivable_report";
13
14     private static InputStream loadResource(String filename) {
15         return App.class.getClassLoader().getResourceAsStream(filename);
16     }
17
18     public static void main(String[] args) throws Exception {
19
20         InputStream reportStream = loadResource(REPORT_FILENAME + ".jrxml");
21
22         Collection<Customer> customers = CustomerProvider.loadCustomers();
23
24         JRBeanCollectionDataSource beanColDataSource = new
25             JRBeanCollectionDataSource(customers);
26
27         Map<String, Object> parameters = new HashMap<String, Object>();
28
29         JasperDesign jasperDesign = JRXmlLoader.load(reportStream);
30         JasperReport jasperReport = JasperCompileManager.compileReport(jasperDesign);
31         JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, parameters, beanColDataSource);
32         JasperExportManager.exportReportToPdfFile(jasperPrint, REPORT_FILENAME + ".pdf");
33     }
34 }
35 }
```

Lines 14 – 16 define a `loadResource` method, which allows us to load a resource (such as the `jrxml` file) from the JAR file.

On line 20, we load the `jrxml` file from the JAR as an `InputStream`. We then load our list of customers on line 22 using the `CustomerProvider` class.

Next, on line 24, we initialize the data source for our report using the `customers` collection, and we initialize an empty hash of parameters on line 27. We'll use the `parameters` hash shortly to pass an image to the report.

Finally, on lines 29 – 32, we load the report from the input stream, compile it, and fill it using the data source we created, along with the empty parameters hash. We then export it to a PDF file in the current directory.

3. Edit the `jrxml` file in `src/main/resources` and, in the `jasperReport` tag at the top, change the `language="groovy"` attribute to `language="java"`. Otherwise, you may get an error when your program attempts to compile the report.
4. Package and run your program. Be sure to run the fat JAR file. It should generate a PDF file for you. Open the PDF and observe that it matches the report we designed in Jaspersoft Studio.
5. Add the following line to the end of the `main` method:

```
JasperExportManager.exportReportToHtmlFile(jasperPrint, REPORT_FILENAME + ".html");
```

6. Package and run your program. It should now output the report in PDF and HTML formats.

## 13 Passing an Image as a Parameter

The last thing we have to fix is the image in the report – it's still just appearing as a red rectangle. Recall that we created a `logo` parameter in Section 8 and we configured the image in the report to use this parameter as its expression.

This allows us to pass an `Image` object as the `logo` parameter, and have that image displayed as the logo.

1. Download `logo.png` from the following Gist and place it in your `src/main/resources` directory:  
<https://gist.github.com/jsuwo/9167132>
2. Add the following code to the `main` method:

```
package ca.uwo.csd.cs2212.USERNAME;

.
.
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

public class App {

    .
    .

    public static void main(String[] args) throws Exception {

        InputStream reportStream = loadResource(REPORT_FILENAME + ".jrxml");
        InputStream logoStream = loadResource("logo.png");
        BufferedImage logo = ImageIO.read(logoStream);

        .
        .

        Map<String, Object> parameters = new HashMap<String, Object>();
        parameters.put("logo", logo);

        .
        .

    }
}
```

Here, we load `logo.png` as an `InputStream` from the JAR file, and then read it in as a `BufferedImage`. We then add the object to the `parameters` hash that is passed to the report, specifying the name of the parameter to be `logo`.

3. Package and run your program. The logo should now appear in both the PDF and HTML reports.

## 14 JasperReports Resources

As with the other labs, we've only scratched the surface of JasperReports in this lab. For more information on it, see the following documents:



- <http://jasperreports.sourceforge.net/JasperReports-Ultimate-Guide-3.pdf>
- <http://community-static.jaspersoft.com/sites/default/files/docs/jaspersoft-studio-user-guide.pdf>
- <http://jasperreports.sourceforge.net/api/>

## 15 Submitting Your Lab

- Do **not** submit the PDF and HTML reports in your repository.
- Commit your code and push to GitHub.
- To submit your lab, create the tag `lab6` and push it to GitHub. For a reminder on this process, see Lab 1.