# CPD3314 Final Project – Data API

Build the Following Project and Submit to Dropbox on or before Apr. 16th, 2015

Your employers have a set of product records that are stored as an XML-formatted dataset. They have been given to you as ORIGINALS.xml. In a flash of foresight, your employer realized that many modern tools prefer other data formats including JSON and YAML. In addition, they would like a tool that formats all of the data into SQL statements, and another tool that formats all of the data into HTML blocks.

There are many ways to implement this, but your lead developer pointed you in the direction of the following tools:

- snakeyaml: https://code.google.com/p/snakeyaml/
- JSON.Simple: https://code.google.com/p/json-simple/
- Simple XML Serialization: http://simple.sourceforge.net/

The following specific behaviours are expected:

## Command Line Arguments

When built and run on the command line (eg- `java CPD3314-Project ...`) the project should accept the following command-line arguments:

| | |
|---|---|
| `-format=<XML\|JSON\|YAML\|SQL\|HTML>` | Formats the output as appropriate: XML, JSON, YAML, SQL, or HTML. If excluded, default is XML. |
| `-sort=<A\|I\|D>` | Sorts the output (A)lphabetically by Name, by (I)D, or by (D)ate. If excluded, default is to preserve the source ordering. |
| `-limit=<#>` | Limits the output to a certain number of lines. Performed last (ie- after sorting and filtering). |
| `-getID=<ID>` | Retrieves a single result by its ID. |
| `-getDate=<DATE>` | Retrieves all results that match a given date. |
| `-find=<CONTENTS>` | Retrieves all results that include the given case-insensitive content in their name or description. |
| `-o=<filename>` | Outputs to a specific filename. Appropriate file extension is added by the program. If filename is excluded, default is CPD3314.<ext> where <ext> matches the appropriate output format. |

## Command Line Examples

**`java CPD3314-Project`**

Produces an XML file named CPD3314.xml of all products, with no particular sorting

**`java CPD3314-Project -format=XML`**

Produces an XML file named CPD3314.xml of all products, with no particular sorting

**`java CPD3314-Project -format=YAML`**

Produces a YAML file named CPD3314.yaml of all products, with no particular sorting

**`java CPD3314-Project -format=JSON`**

Produces a JSON file named CPD3314.json of all products, with no particular sorting

**`java CPD3314-Project -format=SQL`**

Produces an SQL file named CPD3314.sql of all products, with no particular sorting

**`java CPD3314-Project -format=HTML`**

Produces an HTML file named CPD3314.html of all products, with no particular sorting

**`java CPD3314-Project -o=test`**

Produces an XML file named test.xml of all products, with no particular sorting

**`java CPD3314-Project -o=ten -limit=10`**

Produces an XML file named ten.xml of the first ten products, with no particular sorting

**`java CPD3314-Project -sort=A -o=test -limit=10`**

Produces an XML file named test.xml of the first ten products, sorted by name ascending

**`java CPD3314-Project -sort=I -o=test -limit=10`**

Produces an XML file named test.xml of the first ten products, sorted by ID ascending

**`java CPD3314-Project -sort=D -o=test -limit=10`**

Produces an XML file named test.xml of the first ten products, sorted by date ascending

**`java CPD3314-Project -getID=400 -o=test`**

Produces an XML file named test.xml that only contains the product with ID 400

**`java CPD3314-Project -getDate=2014-08-14 -o=test`**

Produces an XML file named test.xml that only contains the products added on Aug. 14, 2014

**`java CPD3314-Project -find=Desk -o=test`**

Produces an XML file named test.xml that only contains the products with “Desk” in their name or description

**`java CPD3314-Project -find=”Rich Mahogany Desk” -o=test`**

Produces an XML file named test.xml that only contains the products with “Rich Mahogany Desk” in their name or description

 

## Data Model

You must implement the following class in your code. It can have many more private methods or data fields, but the following must exist exactly. Unit tests have been established by Quality Assurance to make sure that the code complies with expected behaviours, so that future developers can build against your API.

| Product |
| --- |
| - id : Integer<br>- name : String<br>- description : String<br>- quantity : Integer<br>- dateAdded : String |
| + Product(void) : Product<br>+ Product(id: Integer, name : String,<br>description : String, quantity: Integer) :<br>Product<br>+ getId(void) : Integer<br>+ getName(void) : String<br>+ getDescription(void) : String<br>+ getQuantity(void) : Integer<br>+ getDateAdded(void) : String<br>+ setId(id : Integer) : void<br>+ setName(name : String) : void<br>+ setDescription(description : String) : void<br>+ setQuantity(quantity : Integer) : void<br>+ setDateAdded(date : String) : void<br>+ toXML(void) : String<br>+ toYAML(void) : String<br>+ toJSON(void) : String<br>+ toSQL(void) : String<br>+ toHTML(void) : String |

## Implementation Tips

### Sorting

If you store all of the Product elements in a data structure like an `ArrayList`, then there is a method called `sort(Comparator c)`, which allows you to send a Comparator. Here is a simple Comparator for backwards-alphabetic order (ie- Z→A).

```
private static Comparator<Product> ZToAByName = new Comparator<Product>() {
    @Override
    public int compare(Product o1, Product o2) {
        return (o2.getName().compareTo(o1.getName()));
    }
};
```

You are also welcome to use any other sorting method you like, including storing the data in an XML DOM, or manual implementation of a system like QuickSort, MergeSort, or any other algorithm.

### Searching

The simplest search method is to iterate through the list (ie- use a for-loop) and determine if each element is a match. For algorithms that must touch every element (ie- Search by Contents) this is sufficient and ideal. However, some search methods (ie- by ID or by String) will benefit greatly from sorting the data first and then implementing another search algorithm, like binary search.

### Other Files, Variables, Methods, Classes, etc…

Your starting files include a CPD3314-Project.java file, and a Product.java file. These two classes will be tested against, and must comply with the tests.

Beyond those two files, you can include anything you feel is necessary. There are NO deductions for security holes, or mismanaged permissions, or any other real-world issues. This is a straight "build to specification" project. You can use any tools you want to meet the specifications.

## From ProfRussell: Expectations, Research and Citations

This problem is something that I would reasonably expect a Junior developer to complete in less than a week of full-time hours, on their first day at the job, supported by asking questions of their peers. I would expect a Senior developer to be able to complete this in less than a day, possibly as part of a pair-coding technical interview.

You have six weeks. Please manage your time effectively. This is not a trivial task, and you do have other work to complete. If it takes you all six weeks and you still don't finish everything, that's okay, there will be lots of opportunity to practice before graduating.

It is recommended to do extensive research of the problem before committing to a final solution. While researching, please keep track of any references that you use. If you use a tutorial, or part of a tutorial, make sure to note that in the source code with a comment similar to the following:

```
/* CODE CITATION:
 * Oracle Java Documentation - Reading XML Data into a DOM
 * http://docs.oracle.com/javase/tutorial/jaxp/dom/readingXML.html
 */
```

Developers spend much of their time researching solutions, and it is perfectly normal and expected to seek out references. Especially if you are using a third-party tool like JSON.Simple, or snakeyaml.

## Grading Scheme

| | | | | | |
|---|---|---|---|---|---|
| Project Proposal and Updates | 0%<br>Nothing Submitted | 2.5%<br>One poor-quality submission | 5.0%<br>One high-quality submission, or two poor-quality | 7.5%<br>Two high-quality submissions, or three poor-quality | 10%<br>Three high]quality submissions |
| Coding Style | 0%<br>Nothing Submitted | 2.5%<br>No comments, no structure, no formatting | 5.0%<br>Minimal comments, and basic source formatting | 7.5%<br>Some Javadoc, good source formatting, minimal structure | 10%<br>All Javadoc, good formatting, perfect structure |
| Unit Tests | Up to 80%<br>Extensive testing is provided with the starting files. Your project must pass these tests to achieve the majority of its grades. Some tests are unit (ie- one thing) tests, while others are end-to-end behaviour tests. The percentage of tests you pass will be used to calculate the remaining grade of your project. | | | | |

## Sample Outputs for `toXXX():String` methods:

### toXML()

```
<product>
    <id>1</id>
    <name>vel</name>
    <description>Mauris enim leo, rhoncus sed.</description>
  <quantity>26</quantity>
  <dateAdded>2014-10-14</dateAdded>
</product>
```

### toYAML()

```
dateAdded: '2014-10-14'
description: Mauris enim leo, rhoncus sed.
id: 1
name: vel
quantity: 26
```

### toJSON()

```
{
    "quantity": 26,
    "name": "vel",
    "description": "Mauris enim leo, rhoncus sed.",
  "id": 1,
  "dateAdded": "2014-10-14"
}
```

### toSQL()

```
INSERT INTO Products VALUES (1, "vel", "Mauris enim leo, rhoncus
sed.", 26, "2014-10-14");
```

### toHTML()

```
<div class="product">
     <h1>vel</h1>
     <p>ID: 1</p>
     <p>Mauris enim leo, rhoncus sed.</p>
     <p>Quantity: 26</p>
     <p>Added: 2014-10-14</p>
</div>
```

## Sample Outputs for File Formats (eg- `-format=XXX`)

### XML – Note, all XML output should be wrapped in a <products> root element

```
<products>
    <product>
        <id>1</id>
        <name>vel</name>
        <description>Mauris enim leo, rhoncus sed.</description>
      <quantity>26</quantity>
      <dateAdded>2014-10-14</dateAdded>
    </product>
    <product>
        <id>2</id>
        <name>non quam nec</name>
        <description>Aenean fermentum.</description>
      <quantity>16</quantity>
      <dateAdded>2014-12-20</dateAdded>
    </product>
</products>
```

### YAML – Note, indentation is very important in YAML, do not include extra indentation

```
---
dateAdded: '2014-10-14'
description: Mauris enim leo, rhoncus sed.
id: 1
name: vel
quantity: 26
---
dateAdded: '2014-12-20'
description: Aenean fermentum
id: 2
name: non quam nec
quantity: 16
---
```

### JSON – Note, each Product object is in an array named “products” inside of an object

```
{
    "products": [
        {
            "quantity": 26,
            "name": "vel",
            "description": "Mauris enim leo, rhoncus sed.",
           "id": 1,
           "dateAdded": "2014-10-14"
        },
        {
            "quantity": 16,
            "name": "non quam nec",
            "description": "Aenean fermentum",
           "id": 2,
           "dateAdded": "2014-12-20"
        }
    ]
}
```

### SQL – Note, include the CREATE TABLE line at the start of all SQL Files

```
CREATE TABLE Products (id INT, name VARCHAR(50), description TEXT,
quantity INT, dateAdded DATE);
INSERT INTO Products VALUES (1, "vel", "Mauris enim leo, rhoncus
sed.", 26, "2014-10-14");
INSERT INTO Products VALUES (2, "non quam nec", "Aenean fermentum.",
16, "2014-12-20");
```

### HTML – Note, include the DOCTYPE html, head and body tags to make a complete file

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div class="product">
      <h1>vel</h1>
      <p>ID: 1</p>
      <p>Mauris enim leo, rhoncus sed.</p>
      <p>Quantity: 26</p>
      <p>Added: 2014-10-14</p>
</div>
<div class="product">
      <h1>non quam nec</h1>
      <p>ID: 2</p>
      <p>Aenean fermentum.</p>
      <p>Quantity: 16</p>
      <p>Added: 2014-12-20</p>
</div>
</body>
</html>
```