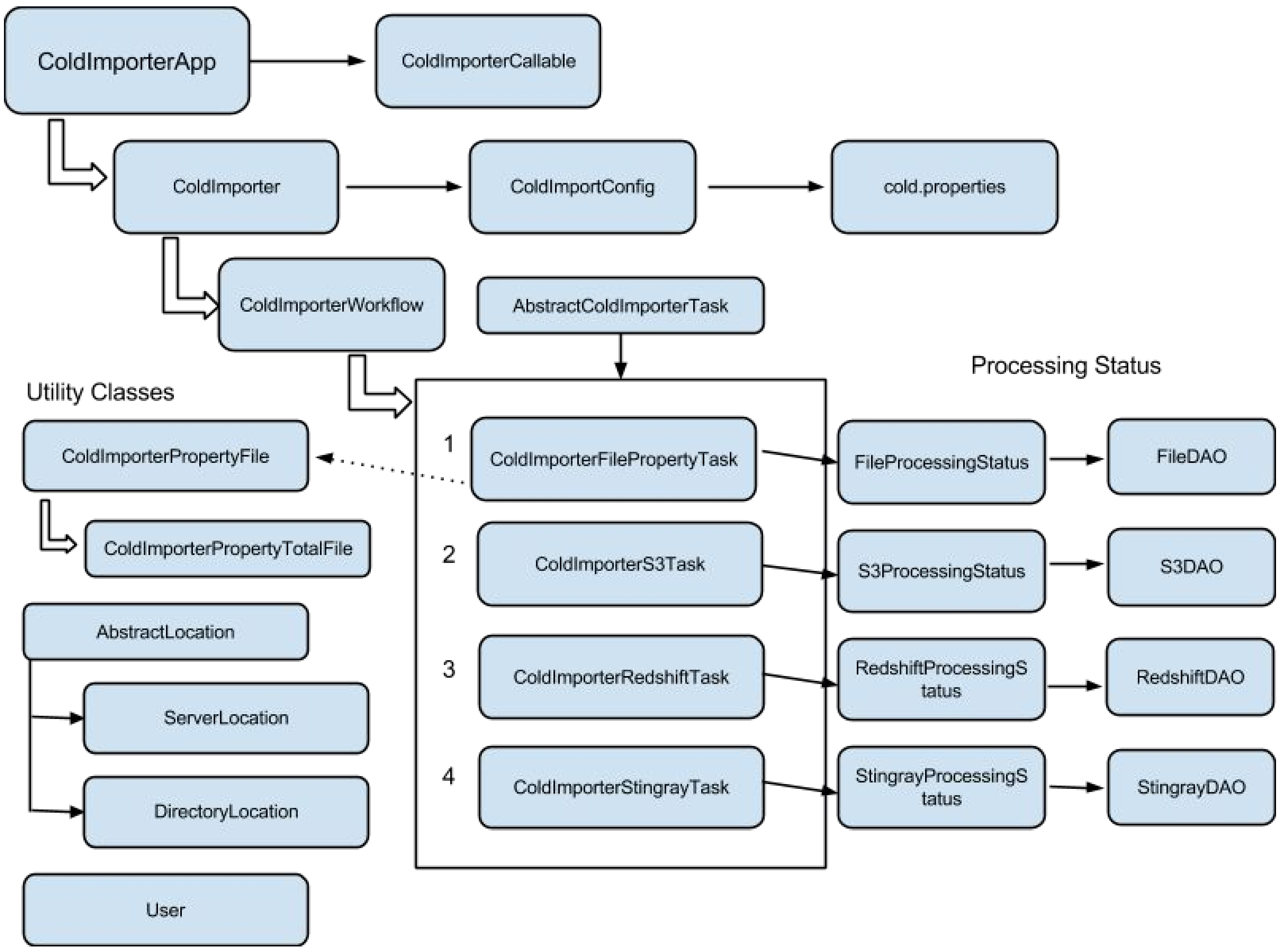
COLD Class Design

Class Diagram



Class Designs

```
ColdImporterApp {
        // contains main method, driver class for the ColdImporter
        // wraps everything in a try catch, runs the process
        // look at DataQuickImporterApp for specific information

}

ColdImporter {
        // actually does all of the configuration for the ColdImporter
        // handles logic to setup Workflow to start process
        // look at DataQuickImporter for reference
        // has init method which instantiates the ColdImporterWorkflow (pipeline manager)

        init() ----> instantiates workflow, calls run

}

ColdImporterWorkflow {
        // manages the pipeline process through the run method
        // run manages the other objects for each component of the process
        // look at Jian's ColdImporter for reference
        // each component has a method that provides functionality

        run() ----> starts workflow process

}

ColdImporterCallable {
        // manages the ColdImporterApp method to allow multi-threading
        // tutorial: http://www.vogella.com/tutorials/JavaConcurrency/article.html#threadpools

}

ColdImportConfig {
        // does all of the config management for connecting to FTP
        // look at DataQuickImporterConfig for example

}
```

```
cold.properties {
        // configuration file for FTP and any other data importer we use
        // look at dataquick.properties as an example

}
```

Workflow Pipeline Classes

```
AbstractColdImporterTask {
        // abstract class for all of the workflow task classes
        // will contain generic information about each task

}

ColdImporterPropertyFileTask {
        // class that encapsulates logic for the downloading and unzipping property files
        // actually has methods to perform the actions of the first task in the workflow
        // should communicate with with FileProcessingStatus class about updating metadata
        // look at ColdImporterUtils corresponding methods:
                downloadFromFTPToLocation and unzipFilesInLocation

}

ColdImporterS3Task {
        // class that encapsulates logic for uploading files to S3 buckets
        // actually has methods to perform the actions of the second task in the workflow
        // should communicate with with S3ProcessingStatus class about updating metadata
        // look at ColdImporterUtils uploadToS3Buckets

}

ColdImporterRedshiftTask {
        // class that encapsulates logic for copying files from S3 buckets to Redshift clusters
        // actually has methods to perform the actions of the third task in the workflow
        // should communicate with with RedshiftProcessingStatus class
        // look at ColdImporterUtils copyFromS3ToRedshift

}
```

```
ColdImporterStingrayTask {
        // class that encapsulates logic for importing files from Redshift clusters to stingray db
        // actually has methods to perform the actions of the final task in the workflow
        // should communicate with with StingrayProcessingStatus class
        // look at ColdImporterUtils readFromRedshift

}
```

Processing Status

For information on the processing status/metadata objects check out this document: https://docs.google.com/document/d/1VrsR-399eR5AxZKy5a9PTpFpKVLA9RoLphenQUMlgAU/edit

Instead of doubling and up and putting the class headers on this doc, I have a very detailed account of the strategy for tracking the processing status of the workflow. It accounts in detail the POJO's class design, table schemas, and DAO classes. This includes:

COLD Importer Property File Processing Status Object
COLD Importer S3 Processing Status Object
COLD Importer Redshift Processing Status Object
COLD Importer Stingray Processing Status Object

General Utility Wrappers

```
ColdImporterPropertyFile extends File {
        // wrapper for property file object that contains:
            -   getters and setters for specific file download
            -   information that can be transferred to processing status object

}
```

```
ColdImporterPropertyTotalFile  {
        // wrapper for extra file in download with metadata about the downloaded file
        // mainly used for veritification that download contains proper contents
        // will be used to update processing status as it’s the initial standard for comparison

        propertyFile: ColdImporterPropertyFile

}


AbstractColdImporterLocation  {
        // generic location object that will be used as a wrapper
        // pevents passing around long strings of locations

        name: String
        path: String

}

ColdImporterServerLocation extends ColdImporterLocation {
        // will see if it’s necessary to make this distinction from DirectoryLocation

        name: String
        path: String
        host: String
        username: String
        password: String
        active: Boolean/enum

}

ColdImporterDirectoryLocation extends ColdImporterLocation {
        name: String
        path: String
        domain: String
        active: Boolean/enum

}
```

ColdImporterUser {

  // wrapper to prevent passing around long strings into method
  // used to keep track of who is updating the properties
  // might need to make this abstract and differentiate types of users

  username: String
  password: String
  awsCreds: BasicAWSCredentials

}

Testing Classes

ColdImporterAppTest {

  // tests that the app starts the ColdImporter and runs on multiple threads
  // Test:

- multiple threads are running different instances of ColdImporter
- ColdImporter runs all the way through with no complications

}

ColdImporterTest {

  // makes sure everything wraps correctly when running a ColdImporterWorkflow
  // makes sure the configuration is complete and a connection it setup with FTP server
  // Test:

- ColdImporterWorkflow instance can be correctly generated and ran
- Configuration is properly wrapped and and connection is complete

}

ColdImporterWorkflowTest {

  // makes sure the Workflow has started and each part of the process was initiated
  // Test:

- ColdImporterWorkflow delegates each task in the proper order
- Logic is working correctly for each task and the process finishes without any errors and new data resting in *stingray*
- Overall Processing Status objects are being created and updated properly at the beginning and completion of each task

}

ColdImporterCallableTest {

// controls the logic for the multiple-threads of ColdImporter running in main class

// Test:

- ColdImporterApp contains multiple threads running instances of ColdImporter
- each thread runs independently and smoothly shares memory with the others

}

ColdImportConfigTest {

// wrapper for configuration methods and properties that need to be set

// Test:

- configuration sets up properly and the ColdImporter is able to smoothly transition to the workflow process
- imports the cold.properties file correctly and other one's if necessary

}

Testing Classes - Workflow Pipeline Classes

ColdImporterPropertyFileTaskTest {

// tests that a PropertyFilePOJO is correctly created upon download and unzipping

// Test:

- does a complete PropertyFile get created upon starting the workflow?
- Is it downloaded and unzipped in the correct directory?
- Is the metadata object correctly updated throughout the process?

}

ColdImporterS3TaskTest {

// tests that a PropertyFile object's data is correctly uploaded to the right S3 bucket

// Test:

- does my data get correctly uploaded to the right S3 bucket?
- Did I get the right confirmation from AWS?
- Did the metadata object update properly with success and failure?

}

ColdImporterRedshiftTaskTest {

// tests that a data file in a S3 bucket is copied to correct Redshift cluster

// Test:

- does my data get correctly copied over to Redshift?

- Did I get the right confirmation from AWS?
- Did the metadata object update properly with success and failure?

}

ColdImporterStingrayTaskTest {

// tests that a data file is imported from a Redshift cluster into *stingray*

// Test:

- Is the data able to be formatted properly for storage in *stingray*?
- does my data get imported into the correct table in *stingray*?
- Did I get the right confirmation from AWS and my own status code?
- Did the metadata object update properly with success and failure?
- Can the newly stored objects have basic CRUD operations performed on them in *stingray*?

}

Testing Classes - General Utility Wrappers

ColdImporterPropertyFileTest {

// makes sure the wrapper object is correctly made from the download

// Test:

- getters and setters for specific file download
- information that can be transferred to processing status object
- information can be accessed from this POJO to the data file, and the data file can actually be manipulated from the methods in this POJO

}

ColdImporterPropertyTotalFileTest {

// makes sure the wrapper object is correctly made from the download total file

// makes sure the the wrapper interacts correctly with the PropertyFile wrapper

// Test:

- getters and setters for specific file download
- information that can be transferred to processing status object
- information can be accessed from this POJO to the total data file, and the data file can actually be manipulated from the methods in this POJO
- this object is correctly representing the corresponding POJO for PropertyFile, and also the total file it objectifies

}

```
ColdImporterServerLocationTest {
        // ServerLocation object correctly gets instantiated
        // Test:
            -   getters and setters for specific server work properly
            -   test formatFilePattern helper method
}
```

Testing Classes - Processing Status

Check out the Processing Status schema doc:
https://docs.google.com/document/d/1VrsR-399eR5AxZKy5a9PTpFpKVLA9RoLphenQUMIgAU/edit and look at the very bottom under the Testing section.