Design Document for Software Solution

# 1. Introduction:

This is a design document to explain the purpose of the solution, every class involved in the solution, the functionalities implemented and offered, design methodologies, system configuration and testing approach.

# 2. Purpose:

The main purpose for this software solution is to develop software to download data from various sources through different protocol. This solution should be extensible, maintainable for future developments. This solution should be capable to handle different types of sources in terms of file size, server speed etc. Additionally, it should support resuming download and will save the file in the final configurable location only if the download is fully completed.

# 3. Packages, Classes and Interfaces

### com.download.main:

This is the main package to initialize and run the software.

### DownloadDemo:

This is the main class to execute the application. This class will populate the system configuration and then read a file “input.xml” to extract a number of source urls. It invokes Context to implement different protocol which handles different source url.

### Context:

This is the Context class which will be used by the DownloadDemo class to invoke the strategy as per the source url’s protocol. The Context class’s behaviour will be based on the protocol it uses to download from each specific url source.

### ConfigurationSingleton:

This is the Singleton instance to be populated with system configuration from a configuration file called “configuration.xml” in the beginning of the application and will be utilised by all of the protocols once they are invoked.

### com.download.protocols:

This package has protocol handlers to download data/files using specific protocols

### Protocol:

This is the base interface for all protocol to support the strategy design pattern. This interface has a single method to be implemented by all inherited classes.

### HttpProtocol:

This class is inherited from Protocol interface and define strategies to handle data download from Http url source.

### FtpProtocol:

This class is inherited from Protocol interface and define strategies to handle data download from Ftp url source.

### SftpProtocol:

This class is inherited from Protocol interface and define strategies to handle data download from Sftp url source.

### ProtocolFactory:

This is a factory class, mainly used to generate different Protocol instances to be used in Context class depending on the input source url’s protocol.

### com.download.helper:

This package has the helper classes to provide common support to protocol handlers.

### FileHelper:

This is a generic helper class to provide multiple file handling functionalities for all protocols. The functionalities include deleting a file, calculating a file size, merging files into a specific target file, file copy etc.

### ParseURLHelper:

This class is a generic helper class to parse any URL string to extract useful information to be utilised by the different protocol handlers.

### Utility:

This is a helper class to provide system utilities such as to determine the available free memory.

# 4. Functionality:

The solution is developed to provide below mentioned functionalities for http url sources and is capable of downloading data through Http, Ftp and Sftp protocol.

### The program should extensible to support different protocols

To add a new protocol, the solution needs a class to implement the strategy for the particular protocol, along with a single entry in ProtocolFactory class to produce an instance of that protocol, when needed to be invoked.

### Some sources might very big (more than memory)

This is handled in HttpProtocol strategy. When an input external source file is larger than available memory, the file is downloaded using multi-part download mechanism in which the actual file is downloaded by parts and then merged together to a target file. Through this process memory issues can be avoided.

### Some sources might fail in the middle of download

This can be a common issue due to interruption or network failure. The HttpProtocol is developed to handle such cases with resume feature. If any http data download is interrupted, the file is saved in a temporary location. Once we attempt to download the file again, the solution will check if the file exists on the temporary location (which indicates that the file is partially downloaded). The solution will start downloading from the next byte and append the rest of the data to that existing file.

### We don't want to have partial data in the final location in any case.

In every http data download, the data is downloaded to a temporary location until the download is completed. Once download is complete, the file is compared with the source file. If they have the same data (i.e, Data is fully downloaded and both source and destination has the same file size), the downloaded file is moved from temporary location to final location. If destination file in the temporary location has less data than source, it is left in temporary location for future resume process.

# 5. Design Methodologies:

### 1. Strategy Design Pattern:

This is a behavioural design pattern. The main solution is based on strategy design pattern in which the class behaviour or its algorithm is changed at runtime as per the object or scenario which is being handled. Here, based on different protocol source, different protocol handler (behaviour) is invoked.

### 2. Factory Design Pattern:

This is a creational design pattern. Different protocol instances are produced by factory design pattern in which we create object without exposing the creation logic to the client and refer to newly created object using a common interface. Here, ProtocolFactory produces different protocol objects and hands it over to Context class using its common interface reference protocol.

### 3. Singleton Design Pattern:

This is also a creational design pattern. To populate the system configuration the solution used singleton design pattern. This pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine and provides a global access point to get the instance of the class. If it is used for system configuration, it is tightly coupled with most of the classes and should be handled carefully. Here, ConfigurationSingleton is populated by the main DownloadDemo class and utilised throughout the application by different Protocol classes.

# 6. System Configuration:

The system configuration is populated through a configuration file called “configuration.xml” at the beginning of the application and thus is available throughout the life cycle of the application with the help of singleton instance. The file has the paths for temporary and final destination folder as configurable property.

# 7. Testing Approaches:

The testing is performed through an input file called “input.xml” which is basically a configuration file with key value pair. The keys in that file are the protocol names and the values are the testing url sources with different protocols for data download. These keys are fetched at run time and hence, the appropriate protocols (behavioural strategies) are invoked dynamically as needed. Additional tests are done by tweaking the code snippets. For example, forcefully changed the available memory size to a lower value than the external file size to test the multi-part download for large files and putting break on the loop while reading from input stream to test the resume feature.

# 8. Appendices:

### A. Initial UML diagram:

### Untitled Diagram.png