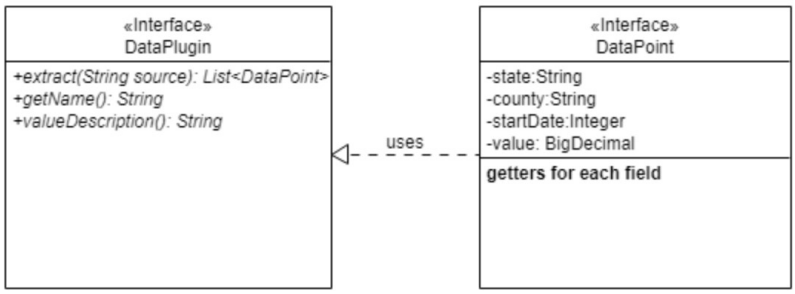
• Framework Domain

Our domain is data with each entry associated with a county in a state of the US and a numerical value. Data plugins simply read tabular data from many types of sources (xls, csv, web) and return an easy-to-use API that returns lists of associated points (e.g., you can query all data associated with counties in state of Florida).

The main purpose of the framework is to allow for a reusable filtering function, which can filter out specific counties, states, and time frames. This API also has library methods that can return analysis of the given data, grouping counties into bigger structures, and different types of analyses (sum, average, stdDev). This data structure is handed off to the Display-Plugin.

• How to Implement Data Plugins



The framework relies on DataPlugins writing the method *extract*, which returns a list of DataPoint type elements to the framework.
The method *extract* can expect a source in the form of a string (e.g., path name, URL) that the framework queried the user for. This framework is in charge of tending to the source of the data and parsing out data entries in the form of DataPlugi
The method *getName* method returns the name to be displayed to the user in the plugin-choosing process.
The method *valueDescription* is the descriptive name of the numerical data being parsed (e.g., poverty percentage) and is queried or given by default by the data plugin.

• How to Add Data Plugins to Project

In the **Team28/plugins** project, you can add your implementation of the *DataPlugin* interface to the package *edu.cmu.cs.cs214.hw5.plugins*

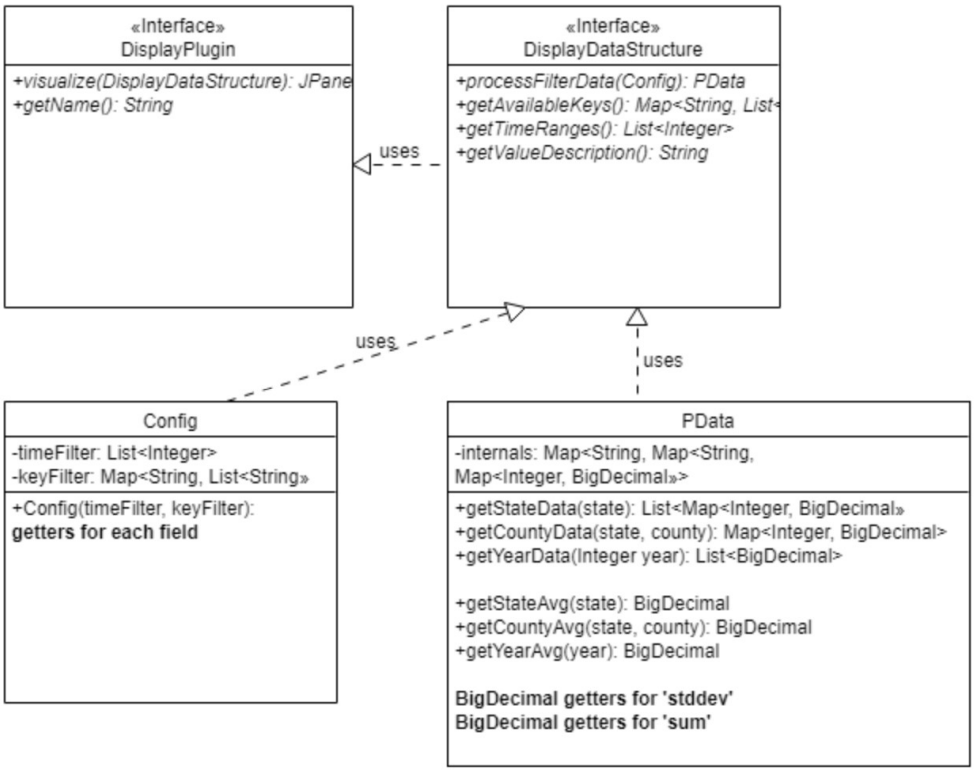
Afterwards, in the same project, refer to the file in

/src/resources/META-INF.services/edu.cmu.cs.cs214.hw5.framework.core.DataPlugin

and add the reference to your file:
e.g. if filename is *XXX*, then add *edu.cmu.cs.cs214.hw5.plugins.XXX*

Your plugin will be automatically loaded by the Framework when *gradle run* is executed.

• How to Implement Display Plugins (& explanation of APIs)



The framework relies on the *DisplayPlugin* to return it a JPanel which it will display in a separate window.

Config

The class *Config* contains two fields, a mapping of states to counties and a list of integer years. The state/county mapping represents which states the plugin writer wants to keep for further analysis, and the list of integers represents which years the plugin writer wants to maintain for analysis.

DisplayDataStructure

The *DisplayDataStructure* is an API that lets the Plugin writer know which times were read (*getTimeRanges)* and which states->counties were read (*getAvailableKeys)* from the source in the last-run plugin. Most importantly, the API functionality *processFilterData* should be called with a *Config* class, to define which of the times and states/counties read-in will be kept for further analysis; this method returns a *PData* object with an analysis-specific API and the data filtered according to *Config*.

The *PData* class has many analysis specific methods. This structure is built from the filtering specifications of Config, and many instances of this object with different filter configurations can be created by the plugin writer for increased functionality.
The *get__Data(…)* methods are the meat of this API, as they return the one-to-one mappings of year->value, which allow the plugin writer to feed these into 3rd-party display APIs like XChart to display charts. The plugin writer can get data from all counties in a state, a specific county in a state, and even data for a year across all states and counties.

The *get__Avg*, *get__Std*, and *get__Sum* methods are the meat of the analysis API, as they return singular BigDecimal values analyzing data from all counties in a state, a specific county in a state, and even data for a year across all states and counties.

DisplayPlugin

The *visualize* method takes in the *DisplayDataStructure* and utilize the APIs from it and *PData* to get the data they require to create a visualization of the data. Given the specificity of the *Config* class, the plugin writer can have user-input regarding what data the user wants to visualize. This user-input also complements the fact that the return type is a Swing JPanel, which allows the plugin writer to add additional functionalities.

• How to Add Data Plugins to Project

In the **Team28/plugins** project, you can add your implementation of the *DisplayPlugin* interface to the package *edu.cmu.cs.cs214.hw5.plugins*

Afterwards, in the same project, refer to the file in

/src/resources/META-INF.services/edu.cmu.cs.cs214.hw5.framework.core.DisplayPlugin

and add the reference to your file:
e.g. if filename is *XXX*, then add *edu.cmu.cs.cs214.hw5.plugins.XXX*

Your plugin will be automatically loaded by the Framework when *gradle run* is executed.

• Demo

We'll use the Web-Scraping-XLS data plugin and data from https://catalog.data.gov/dataset/fy2015-vha-enrollees-by-county to display it on Filterable Display Plugin