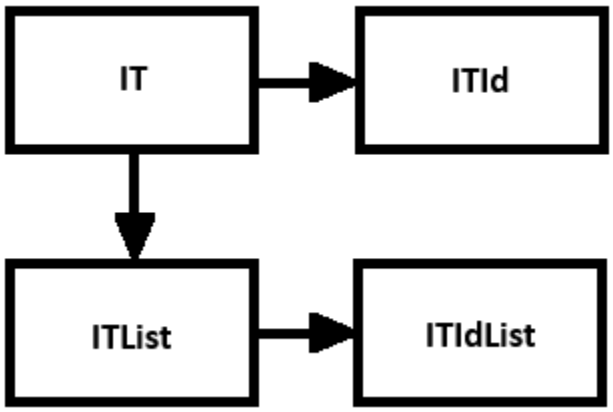
This project (TickersToCandles) simulates the reading of tickers and the creation of candles for different assets. The input/output of this routines takes place through reading/writing files.

This document is divided in two parts: The classes that deal with our purpose are inherited from some base classes that are a part of my personal system design. I practice those for years and they have the specific goal of narrowing down every structure of a project to a few simple standardized concepts. With this in mind it is possible to build highly scaled applications since most of its structures are designed in a simple common pattern. I used to manage a 1.8 million code platform with 1800 tables and I would address any routine by heart in 90% of the times, because 90% of the system were built in this simple yet powerful pattern.

Please note that the creation of this layer affects performance so it cannot be used in parts that need to be really fast, such as container structures withing STL library in C++. It also has a lot of polymorphism practices and dynamic type casting, which can considerably slow down execution of critical processes. However, in most environments that do not demand high performance data management, this pattern usually gets the job done very nicely.

The base pattern



IT – base class for everything that it exists. Everything is an **"IT".**

- Properties
  - **Owner** – represents the class who owns **"IT".**

ITList – represents a list of **“IT’s”.** Notice that it also descends from **“IT”**.

- Properties:
  - **(Owner)** – inherited from **“IT”** class
  - **Add** – Adds a new empty item
  - **Delete**(const int pIndex) – Delete an Item of a given index
  - **Count** – Returns the quantity of items of the list
  - **Clear** – Empties the list
  - **GetItem**(const int pIndex) – returns the specific Item of a given index

ITId – It has **“ID”** and **“Name”** properties.

- Properties:
  - **(Owner)** – inherited from **“IT”** class
  - **ID** – initialized with -1
  - **Name** – Initialized with “”

ITIdList – represents a list of **“ITId’s”.** Notice that it descends from **“ITList”,** which descends from **“IT”.** Everything is an **“IT”.**

- Propeties
  - **(Owner)** – inherited from **“IT”** class
  - **Add, Delete, Count, Clear, GetItem** – inherited from **ITList**
  - **ById** – returns the Item of a given Id
  - **ByName** – returns the Item of a given name

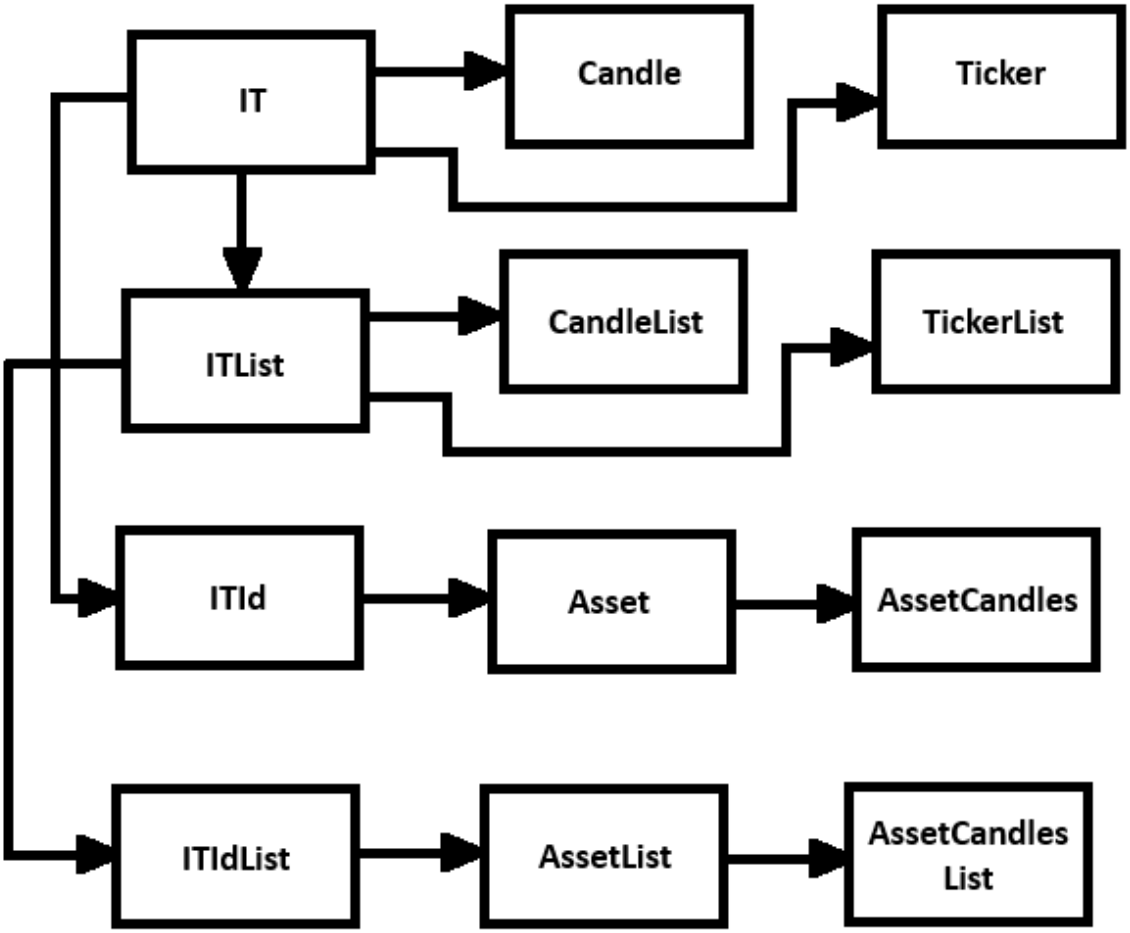
This base class uses vectors to hold a list of pointers to **“IT”** descendent classes. Further version will reely on maps to fast access to Id’s and names.

Every **IT** has an owner who represents the **“ITList”** class that created it. That means every class is connected to every class. This creates an **“IT”** tree binded by its owners and by its items. Therefore, for each class it is really easy to access information present in external classes.

Since everything is an **“IT”** it is possible to create ancestral functions that know descendants’ attributes through **RTTI** info. Because of this, a single routine in a base class can create and parse JSON files, for instance.

Additional details of those patterns are provided in source code comments.

Tickers to Candles Classes



- Every class is ether descendant of **“IT”** or **“ITId”**
- **Candle** Class **(:IT)** holds information about OHCL and volume info.
- **CandleList** Class **(:ITList)** is a list of candles.
- **Ticker** Class **(:IT)** holds information about the ticker (timestamp, key, value)
- **TickerList** Class **(:ITList)** is a list of Tickers.
- **Asset** Class **(:ITId)** has the ID and the Name of an Asset.
- **AssetList** Class **(:ITIdList)** is a list of Assets.

- **AssetCandles** Class **(:Asset)** has 2 Candle List variables: (5 second timeframe and 1 minute timeframe)
- **AssetCandlesList** Class **(:AssetList)** is a list of AssetCandles.

Brief explanation of this routine:

There is a class named **TickerToCandleCtrl** that has as members:

- **TickerList** (List of Tickers)
- **AssetCandlesList** is a list of **AssetCandles**, and it also have 2 **CandleLists** to represent the sum of all **AssetCandles** Candle Lists of their Items.

When **TickerToCandleCtrl.ExecTickersToCandles** is executed:

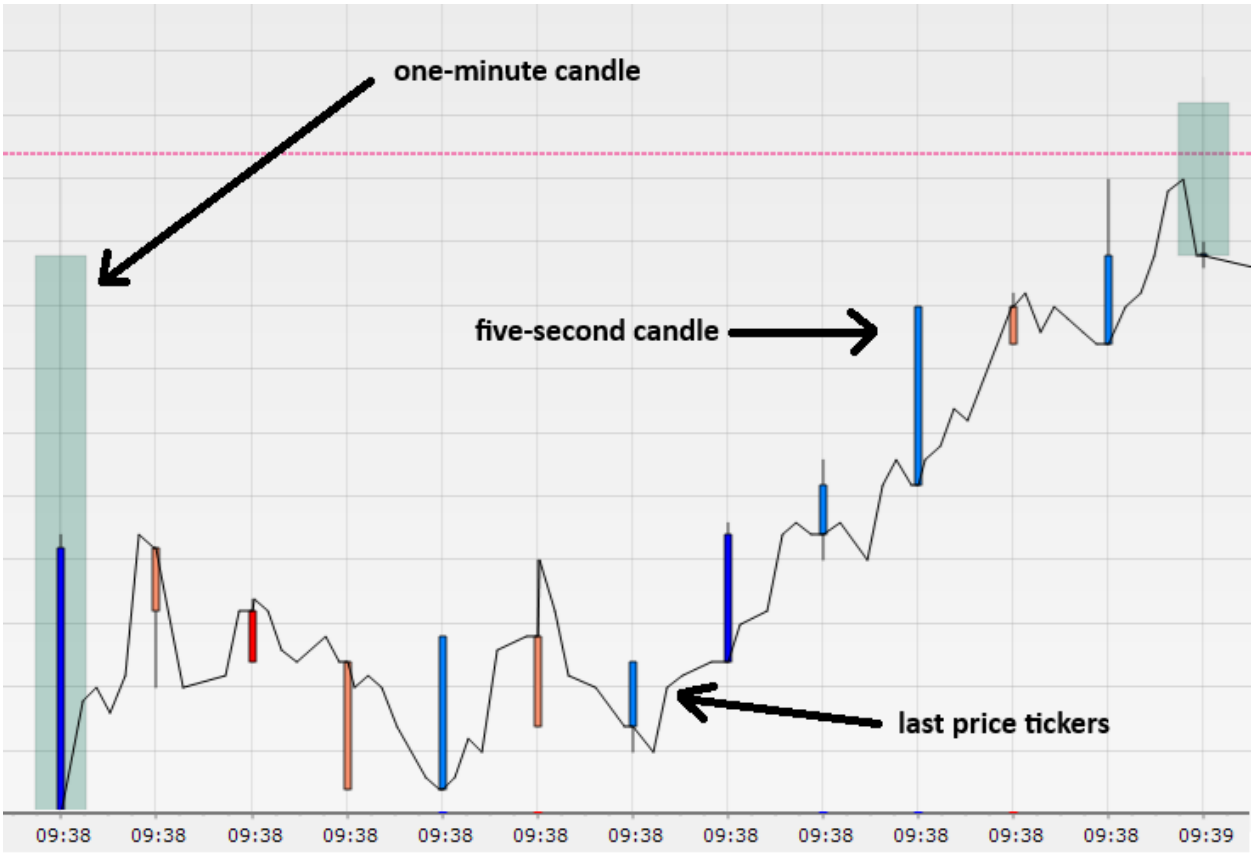
- It reads a file named tickers.csv (in the same directory of this application) and stores its information in memory.
- Then a loop is executed to read each line stored in memory.
    - It retrieves the asset name from this line.
    - It creates a new item (or access a previously created one) in AssetCandlesList according to this asset name.
    - It executes **AssetCandles.ComputeTicker** to pass the ticker information for the respective Asset.
    - This method creates a new item in the “5 second candle list” according to its timeframe, then it fills it with OHCL and volume info.
    - After this process we have something like this:
        - **AssetCandesList**
            - **AssetCandles Asset I**
                - **5S CandeList**
                - Candle 09:30:00 – 09:30:05
                - Candle 09:30:05 – 09:30:10
                - Candle 09:30:10 – 09:30:15
                - Candle 09:30:15 – 09:30:20
                - …
                - **1 Min CandeList (Empty)**
            - AssetCandles Asset II
                - ---
            - AssetCandles Asset III

        - ---
    - AssetCandles Asset IV
        - ---
- **AssetCandles.Create1mCandles** is executed to fill **"1 min Cande Lists"** for each item. It is much faster to build 1 minute candles from 5 second candles than from Tickers itself.
- After this process our tree info looks like this:
    - **AssetCandesList**
        - **AssetCandles Asset I**
            - **5S CandeList**
                - Candle 09:30:00 – 09:30:05
                - Candle 09:30:05 – 09:30:10
                - Candle 09:30:10 – 09:30:15
                - Candle 09:30:15 – 09:30:20
                - …
            - **1 Min Cande List (Empty)**
                - Candle 09:30:00 – 09:31:00
                - Candle 09:31:00 – 09:32:00
                - Candle 09:32:00 – 09:33:00
                - Candle 09:33:00 – 09:34:00
                - ---
- Once we have 5-second and 1-minute candles list for each asset, a final method is called (**FeedMainCandles**) to fill the **MainCadlesList** variables which are members of **AssetCandlesList** Class. This list receives the items of those candle list as pointers, and it does not take ownership of them.
- Methods to save these files as CSV inside "output" folder are executed. This folder structure looks like this:
    - **Output**
        - File allcandles_5s.csv (five-second candles)
        - File allcandles_1m.csv (one-minute candles)
        - **Folder AAPL**
            - AAPL_candles_5s.csv

- AAPL_candles_1m.csv

- **Folder COIN**
  - COIN_candles_5s.csv
  - COIN_candles_1m.csv

- These files support the data represented below:



Additional details of those routines are provided in source code comments.