project design

P3+J3^u! Holdings

Table of Contents

1. purpose:
   1. This document describes the requirements for the main project of P3+J3^u! Holdings. That project, referred to hereafter simply as “the project,” is an application to enable increased consistency and profitability in trading stocks and options. The project automates the functions of trading, with the intention that the code will perform day trading.
2. project summary:
   1. The project utilizes technical indicators as a primary method for determining profitable market positions. Technical analysis is applied repetitively to the stock analyzed as the program runs during the trading day. When the analysis indicates a reversal is occurring, the application will place appropriate orders to update the configured account’s position to best profit from the market’s action. The technical analysis is performed using parameters based on back-testing:

* each stock under analysis is back-tested using a wide range of weights (i.e. 5,6,7,etc.-day moving average crossover with 30,31,32,etc.-day moving average);
* back-testing results are used to select the most profitable weights;
* analysis using the weights occurs for the stock at every *N*th interval throughout the trading day/week/month (as configured);
* each time analysis is performed, the current market position is inspected alongside the technical analysis results;
* if the current market position is not healthy, market orders are created to improve the health of the account’s position;
* at a regular interval (daily/weekly/monthly as configured), a. and b. are performed to update the analysis
  1. The project is coded in an Object Oriented style intended to enable extensibility and maintenance. All code extends a base object, zerg.h. This universal superclass includes methods to perform reflection, including knowledge of its class name, creation of a UID at instantiation, and knowledge of all memory locations the class allocates. The universal superclass also contains methods to perform secure memory release of all memory allocated by the class in which the memory space is overwritten with zeroes (“zeroed”) before release.
  2. Test coverage for the project is provided by Test Driven Development, in which unit tests are written prior to the code they will test. The general rule is that each method should have multiple cases which test expected input and multiple which test invalid input. It’s possible that some methods will not be testable, and that others may require a far broader battery of tests.

It is not necessary for all of the tests to be envisioned and written prior to writing the method; rather, at least one test should be written that will fail until the method is correctly implemented. The remaining tests described above should be functioning before the method is merged to the master branch.

* 1. The project uses its OO nature for security purposes, except insofar as external libraries are used. Although the early iterations of the project are not expected to do much more than wrap vanilla classes supplied by C++ libraries, all objects used in the project should be members of the project.

NO EXTERNAL CLASSES MAY BE INSTANTIATED.

This enables the classes to later implement management of all data used by the class; for example, if the string class simply freed a char[] on deletion but did not zero out the contents, the zString class would instead use an internal char[] for all operations and its memory would be cleared by calling its other superclass’s (zerg) destructor. If the string class stores its contents in clear text, the zString class encrypts its contents and decrypts them on access when the correct key is available. Such a key would be generated from data only knowable to the OS (preferably the security kernel) and supplied to the application on request, rather than stored in the application’s memory.

1. class framework:

The class uses a series of base classes which are designed to permit introspection and dynamic coding. All of the classes used by the project must extend one of these base classes. Each class only accesses its variables using the design constraints required by the superclasses from which it inherits. The project uses a hierarchy of classes extending from one core superclass. Base classes extend this core superclass to enable a variety of functionality; a “list” class enables subclasses to manage their variables using a built-in dynamic list, for example. A “template” class then extends the base class and uses the list class to manage all of its members. It is from this template class that all other classes derive.

* 1. base: The Ultimate Super-Class

The base class contains data to uniquely identify the class, including class name and a UID.

* 1. list: The Array

The list class contains a very flat B-Tree of either primitive types or “base” type objects.

It may or may not be sorted for fast access (as opposed to being sorted by index, a default value assigned iteratively to each new element). It supports a native sort() method that will sort data for any allowable type using a default criteria, as well as a sort(function F) method that takes a function as an argument and applies it to the data.

Traversal: All members can be accessed by an index, like an array. All members can be referenced using a single member and the next() and previous() methods, as in a doubly-linked list. Members can be pushed(), popped() or peeked() as if in a FILO queue or, if an internal variable is set, as if in a FIFO queue.

The number of children per node *TODO: is/isn’t dynamic. If it is static, list the number of children each node has (128? 1024? Why?). If dynamic, list when the B-Tree chooses to flatten itself and vice versa.*

* 1. template: The Direct Super-Class

The template class extends the base class and contains a public list and a protected list. These lists will contain the vast majority of every subclass’s variables (and private variables should be contained in a private list that the subclass creates). Exceptions to this dynamic include temporary variables of limited scope, which should always be of a type which extends the base class and will therefore be securely deleted.