**National University of Computer & Emerging Sciences**

**Karachi Campus**



**Project Report**

**Object Oriented Programming**

**CS-217**

**Section: E (CS)**

**StockEx**

**Group Members:**

19k-0150 Affan Naeem

19k-0321 Shayan Pasha

19k-1410 Wahaj Ameen

**StockEx**

**Introduction:**

StockEx is a stock trading platform that allows users to buy, sell and invest in stocks and securities in the free market.

**Problem:**

 From a user-oriented perspective, the goal with StockEx was a to develop an easy to use application with a very strong backend that ensured no errors occurred in the financial transactions. The main problem was data storage, security and validation as any errors in data could lead to catastrophic consequences.

**Motivation:**

As developers, the reason behind choosing StockEx as our project was to gain experience in a key OOP area – object relation and interaction. Simultaneously, we also sought to gain valuable experience in data storage and database management using SQL and simplistic GUI design with Qt.

**Methodology**

**Database Management:**

For database management, we opted to use the C library implementation of sqlite3. The drawback here was that being implemented in C, sqlite3 makes no use of OO principles or useful OOP features such as exception handling. To remedy this, the first class we created was SQLite, a singleton class that serves as a lightweight wrapper for the sqlite3 C library, providing it with OO capabilities. We also developed SQLiteEx, an exception class that inherits from std::exception and is used in the project to communicate and handle SQLite errors.

**Object-Database Interaction:**

The next step was to develop a way for our objects to interact with the SQLite class. We decided to set a standard pattern for all classes that would require database functionality. This was implemented using an abstract interface class – IRecord. IRecord has a set of 4 implemented functions that provide its derived classes the 4 basic database operations – insert, select, update and delete. To implement these, IRecord makes use of a host of pure virtual functions. This ensures that all data management throughout the project follows a common standard set by IRecord and also encourages code reuse with inherited functions.

**IRecord Derived Classes:**

Across the codebase, a total of 7 classes are derived from the IRecord interface class. Each of these classes maps to a separate table in the database and implements a full functionality of database communication. The common pattern of IRecord inheritance allows the data control classes – primarily RealTime and User – to interact with all database objects easily, regardless of their concrete type. Object interaction is not restricted to only these classes however, as database objects often use the functionality of each other. A good example of this is this code snippet showing the interaction of User, Listing and Share:

**Data Control Classes:**

Primarily, each database object has complete control of its data. However, to facilitate interaction and communication between database objects, we made use of the singleton design pattern to create the monolithic class RealTime. The chief purpose of RealTime is to serve as a data holder during program execution as database details are loaded into program memory. It also provides a uniform method of data access which is used throughout the program to implement object communication and interaction. In true OO fashion, very little of the processing occurs in RealTime as it is not itself the primary owner of the data. Instead, the data processing if offloaded to each database object. This ensures data encapsulation and safety throughout the program.

A secondary data control class is User, which itself is housed in RealTime. The purpose of User is to hold data related to a particular user and implement interaction between different users, such as during stock transactions.

**Helper Classes:**

To ease interaction between SQLite and different database objects, we implemented two helper classes, Condition and ID along with some global functions in Util.hpp.

* **Condition:**

This class is used to provide uniform condition building capabilities that are used in SQL queries.

* **ID:**

A convenience class used by database objects which do not implement primary keys but rely on the Autoincrement feature of sqlite3. Class ID ensures that the correct ID is selected from the database when such objects are created.

* **Util.hpp**

Util.hpp provides an exception throwing macro and two template functions that are used throughout the project to build SQL queries.

**Exception Handling:**

The backend greatly follows the nothrow design pattern – large portions of the backend code are declared noexcept and most exception handling is deferred to the front end. This follows the philosophy of keeping self-generated errors to a minimum. Further, the Expects macro from Util.hpp is used to implement uniform exception throwing throughout the project.

**(Lack of) Data Flow:**

Encapsulation is enforced throughout the codebase – every object assumes complete ownership of its data. While there is unfettered data access in the project, it is read-only. All data processing and data interaction between objects is done by invoking methods instead of direct data access.

**Standard Library:**

In the spirit of code reuse and standardization, the project makes extensive use of the C++ standard template library, in particular its data structures and algorithms. std::vector, std::map, std::set and std::list are used where appropriate. std::shared\_ptr is also used to share database object instances across data control classes.

**Code Optimization:**

C++’s compile time optimizations are used to the maximum by declaring all eligible functions const and noexcept. Static variables are also used where appropriate to avoid multiple constructor and destructor calls. To avoid unnecessary copying, function parameters are passed by reference where possible. Runtime polymorphism with the virtual keyword is also kept to a minimum, leveraging C++’s optimized static compile-time linking.