IMPLEMENTATION OF STOCK TRADING SIMULATOR

TEAM B'S OBJECT-ORIENTED PROGRAMMING PROJECT

2025. 12. 09

Table of Contents

Project Overview

* 페이지내 인물사진은 샘플용 이미지입니다.

Stock Trading Simulator

- OOP term project implementation
- Based on Team A's design specifications
- Console-based stock trading simulation program

Team Information

- Team B Members:
- Kim Da-in, Kim Jin-kwang, Shin Ba-da (Leader)
- Lee Ho-jun, Lim Chan-hyung, Jung Seung-a

Key Features

Diverse Asset Trading

- Spot stocks
- Futures with leverage
- Custom ETFs

Real-time Price Simulation

- Random price fluctuations
- High/low price tracking
- Change rate calculations
- Top movers display

Risk Management System

- 80% loss forced liquidation
- Futures expiry auto-settlement

Implementation Statistics

Classes

19 classes total

- 2 abstract classes
- 5 UI classes
- 12 domain classes

Organized in 3-layer architecture

Methods

Over 70 methods

- Encapsulated logic
- Clear interfaces
- Polymorphic design

Focused on single responsibility

Code Lines

1,114 lines of code

- Clean implementation
- Minimal redundancy
- Single-file implementation

Efficient and readable

Repository

GitHub Repository:

github.com/Casi-a/stock-trading-sim

Complete source code and documentation available

3-Layer Architecture (1)

UI Layer

- Screen (abstract class)
- Base class for all UI screens
- Defines show() virtual method

- Derived classes:
- MainScreen, AllStocksScreen
- HoldingsScreen, TradeScreen
- CustomETFScreen

Domain Layer

- Instrument (abstract class)
- Base for all financial products
- Defines updatePrice(), recalcPrice()

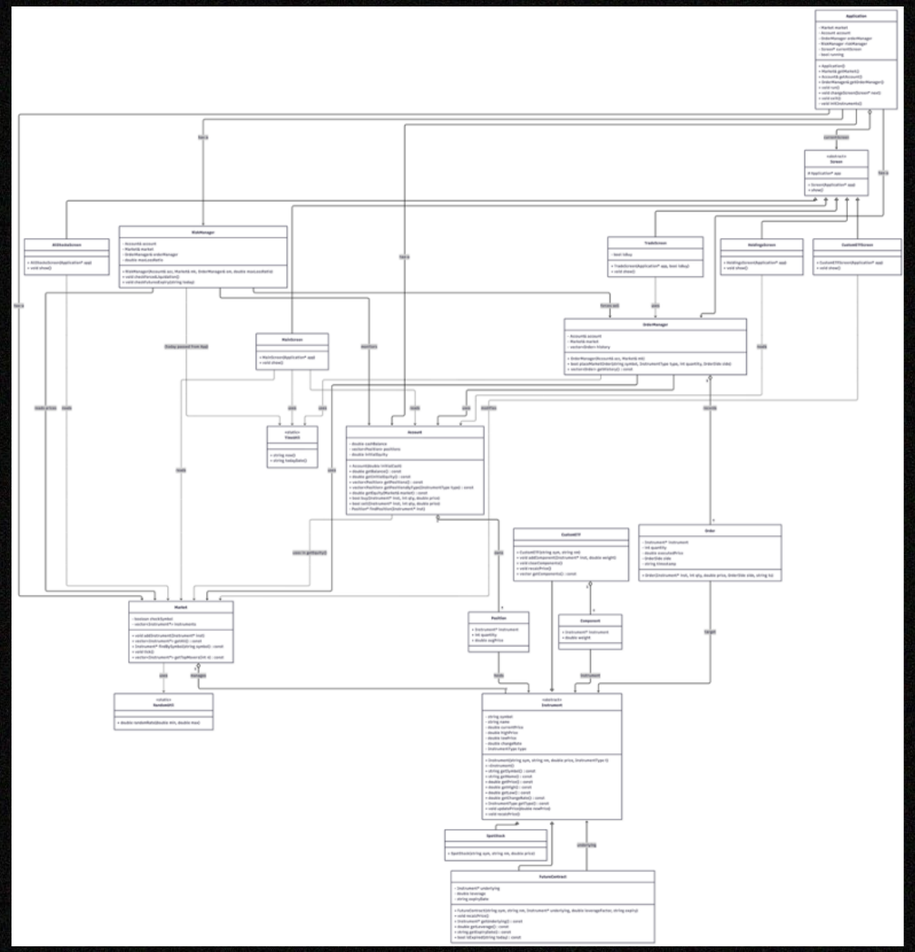
- Derived classes:
- SpotStock: regular stocks
- FutureContract: leveraged derivatives
- CustomETF: user-defined portfolios

- Market: Manages all instruments
and simulates price changes
- Account: Tracks cash balance and positions
- Position, Order: Core data structures
- OrderManager, RiskManager: Handle trading logic

Utility Layer & Core Relationships

- TimeUtil, RandomUtil: Support classes
- Key Relationships:
- Inheritance: Instrument ← SpotStock, FutureContract, CustomETF
- Composition: Market → Instrument*, Account → Position*

3-Layer Architecture (2)



3 Sentence Summary

1. One base class → Three asset types (Polymorphism)
2. Market + Account → Data management
3. Screens + Managers → UI + Logic separation

Class Details - Domain Layer (1)

Instrument

- Abstract base
- Common fields
- Virtual methods

SpotStock

- Regular stock
- Inherits from Instrument

FutureContract

- Leveraged
- Has expiry
- Underlying

CustomETF

- User-defined
- Components
- Weight ratios

Class Details - Domain Layer (2)

Market

- Central repository for all instruments
- Methods:
- addInstrument()
- findBySymbol()
- tick() - simulates price changes
- getTopMovers() - returns top performing instruments
- initInstruments() - loads initial data

Account

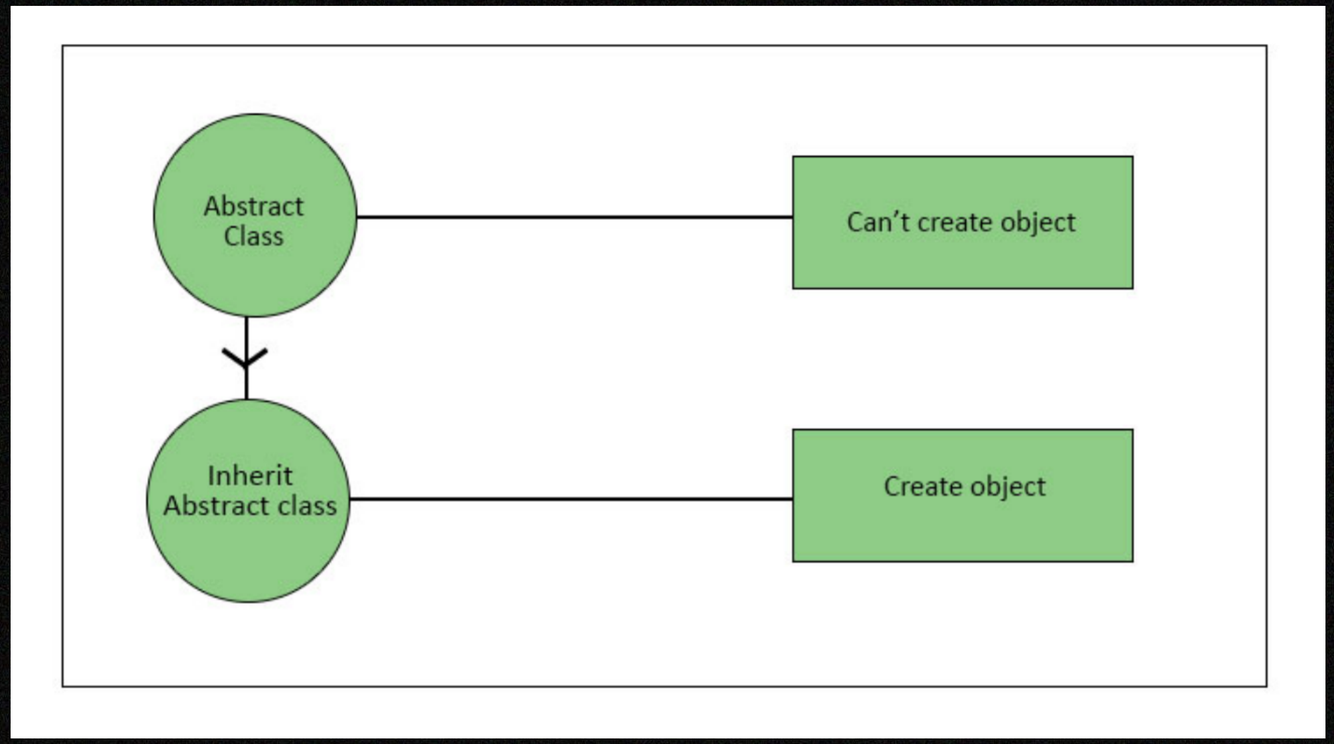
- Manages cash balance and positions
- Methods:
- buy() - creates buy orders
- sell() - creates sell orders
- getTotalEquity() - calculates total value
- getPositionsByType() - filters positions by instrument type

Risk Management

- OrderManager:
- Processes buy/sell orders
- Maintains transaction history
- Validates order parameters

- RiskManager:
- Forces liquidation at 80% loss
- Auto-closes expired futures

Class Details - UI Layer



* 페이지내 인물사진은 샘플 이미지입니다.

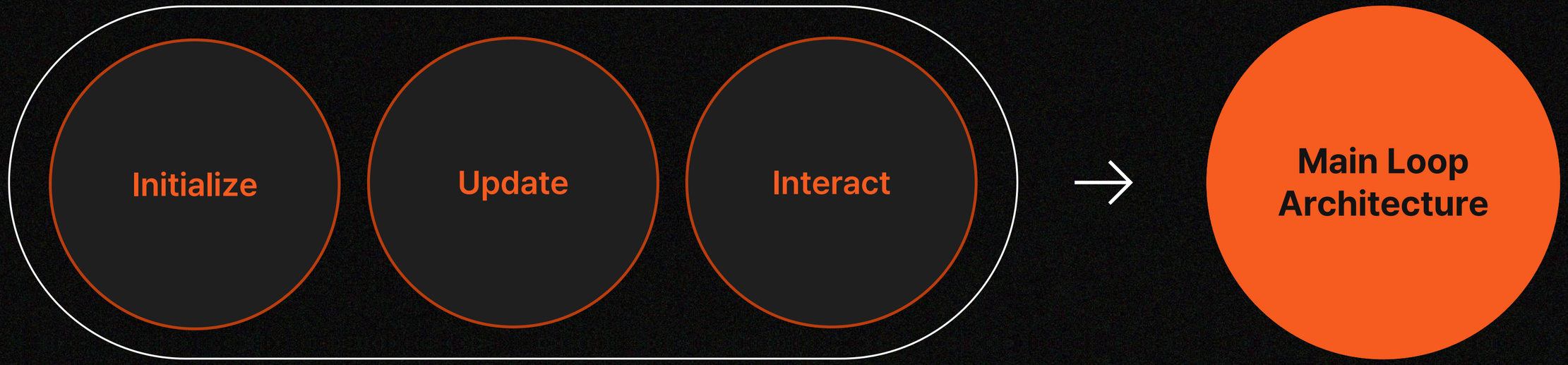
Screen Abstract Class

The Screen class serves as the base for all UI components in the system. It defines a pure virtual show() method that must be implemented by all derived classes, ensuring a consistent interface for displaying content to users.

Derived Screen Classes

Five specialized screen classes inherit from Screen: MainScreen displays market overview, AllStocksScreen shows all available instruments, HoldingsScreen presents user positions, TradeScreen handles buying/selling, and CustomETFScreen manages ETF creation.

Program Execution Flow



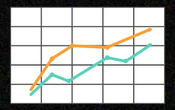
Initialize: Load 50 instruments
Update: Price tick + Risk check
Interact: Display screen +Handle input

Initial Configuration

Capital

- Starting balance: 1,000,000 KRW

- Used for trading all instruments



Stocks

- 50 hardcoded stock symbols

- Various sectors and price ranges

- Random price fluctuation

Futures

- 5 futures contracts

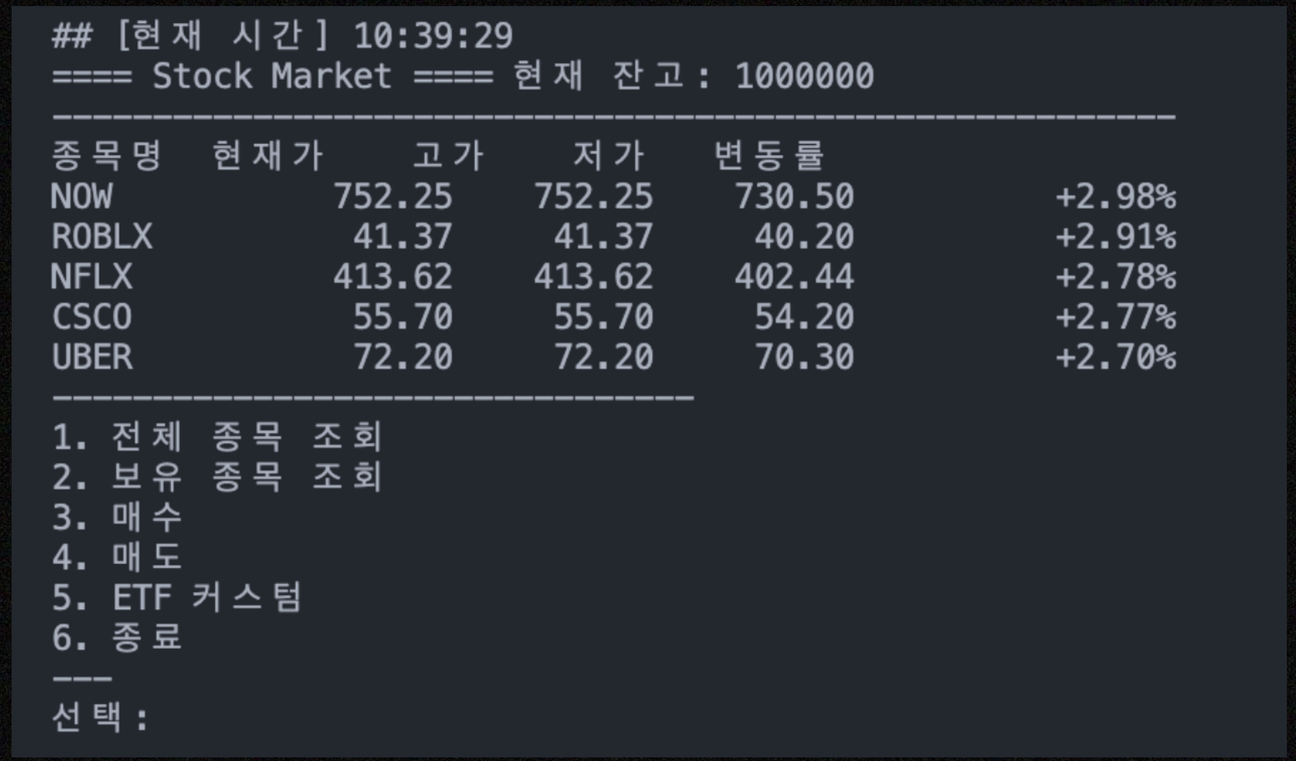
- Based on major tech stocks: AAPL, TSLA, MSFT, NVDA, AMZN

Expiry Date

- All futures expire on 2025-12-31

- Auto-liquidation on expiry date

Main Screen UI



User Interface Design

The main screen displays real-time market data with a focus on top performers. Users can view their current balance and access all system functions through a simple menu-driven interface.

Console-Based Interface

- Top 5 stocks by performance rate
- Current account balance display
- Six menu options for navigation
- Clean terminal-style layout with color-coded price changes

Menu Functions (1)

View All Stocks

- Calls Market.getAll() method
- Displays complete list of 50 stocks
- Shows symbol, name, current price
- Includes high/low prices
- Displays change percentage
- +/- signs for price changes

View Holdings

- Segregated by instrument type
- Separate sections for spot stocks
- Dedicated area for futures
- Custom ETF holdings section
- Shows quantity and average price
- Displays current price and P/L
- Calculates profit/loss percentage

Buy Order

- Select instrument type first
- Choose from spot/futures/ETF
- Enter stock symbol/ticker
- System validates symbol exists
- Input desired quantity
- Confirms available funds
- Executes order and updates position

Menu Functions (2)

Sell Order

- Similar flow to buy process
- Select instrument type first
- Enter stock symbol to sell
- Input quantity to liquidate
- System validates ownership
- Prevents overselling positions
- Shows error if quantity exceeds
- Updates account after execution

Custom ETF Creation

- Enter new ETF symbol and name
- Add component instruments
- Specify weight percentage for each
- System validates total equals 100%
- Creates CustomETF instance
- Registers with Market object
- Available for trading immediately

Exit Program

- Sets running flag to false
- Cleans up allocated memory
- Displays goodbye message
- Terminates application cleanly

Program Demo

Compilation

- g++ -std=c++11
- main.cpp
- -o stock_trading_sim

Execution

- ./stock_trading_sim
- Runs in terminal
- Console interface

Demo Flow

- Main screen
- View all stocks
- Buy AAPL shares
- Check holdings
- Create custom ETF
- Buy ETF units
- Monitor P/L

Live Demo

- Live demonstration
- Real-time simulation
- Q&A session after

Team A Design Review - Strengths

Polymorphic Design

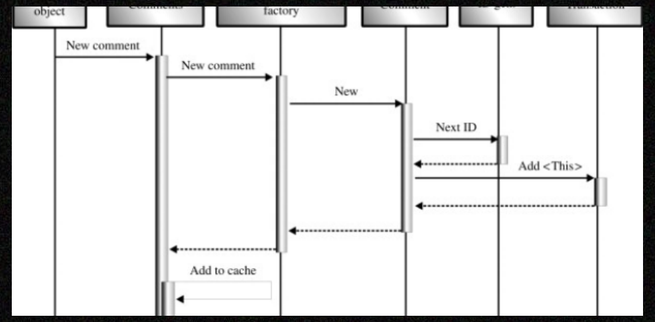
- Instrument base class
- Unified management
- Polymorphic methods
- Type-agnostic lists
- Extensible framework

Flexible Architecture

- Future contracts with flexible underlying
- Supports ETF futures
- Instrument as base
- Adaptable structure

Clear Class Separation

- Screen abstraction
- Application manages current screen
- Clean transitions
- Separation of UI and business logic



Sequence Diagrams

- Detailed interactions
- Clear process flows
- System behavior visualization
- Implementation guide

Team A Design Review - Limitations

Diagram Inconsistency

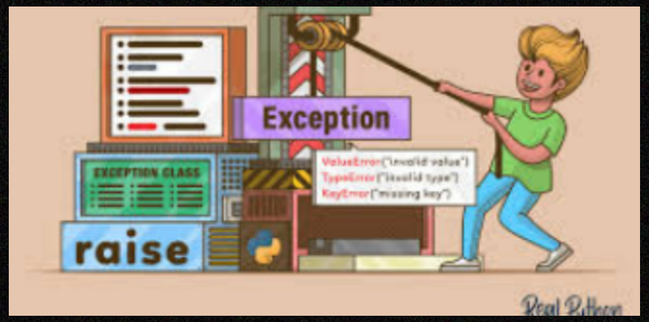
- 7 methods missing
- Class vs sequence diagrams mismatch
- Implementation gaps
- Integration issues

Missing Fields

- FutureContract missing leverage
- Underlying asset reference absent
- Implementation challenges

SRP Violations

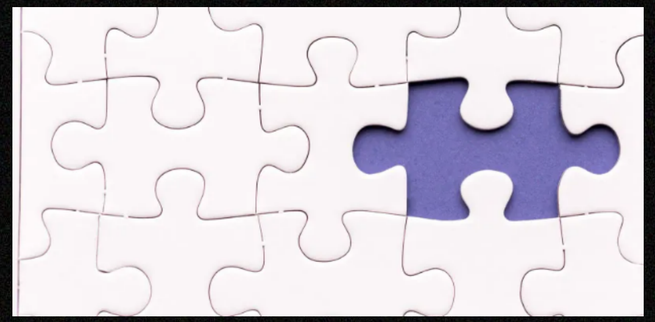
- Account class overloaded
- Buy/sell directly in Account
- OrderManager underutilized



Exception Handling

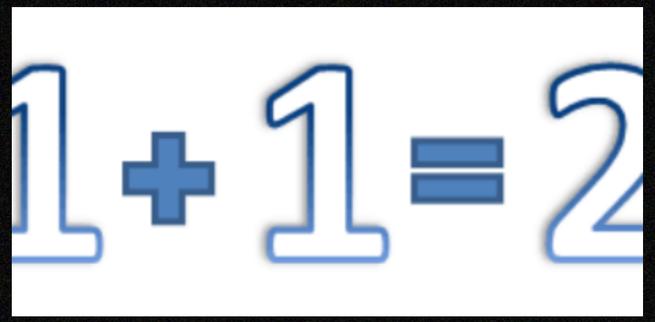
- No error flows
- ETF weight validation
- Invalid symbol handling
- Insufficient funds scenarios

Improvements Made by Team B



Missing Methods

- Implemented all 7 missing methods
- Market:tick(),getTopMovers()
- Application: changeScreen()
- FutureContract: isExpired()

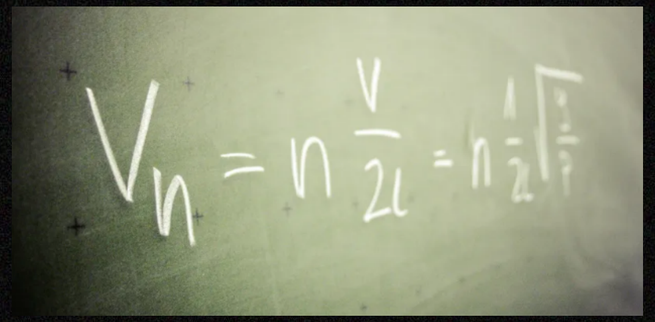


Field Additions

- Added leverage and underlying
- Fixed FutureContract class
- Enhanced inheritance structure

Component Design

- Simplified Component structure
- Used STL pair instead of custom struct



Position Calculation

- Implemented average price calculation logic
- Weighted average formula

What Team Members Learned

OOP Concepts	Practical application of inheritance and polymorphism through Instrument class hierarchy
Design Patterns	Composite pattern in CustomETF and State pattern in screen management
Design Importance	Detailed diagrams are crucial for successful implementation and team alignment
Modular Collaboration	Header/source separation and interface standardization minimized development costs
Smart Pointers	Used shared_ptr for safe reference management of simultaneously accessed objects