

한 권으로 끝내는 주식 자동매매 프로그램 만들기



지은이 : 강준영 외 1명

최종 편집일시 : 2018년 8월 15일 12:05 오후

책 주소:

<https://wikidocs.net/17596>

소스 저장소:

<https://github.com/WooJongSeon/JSWoo-AutoStockTrading/blob/master/SourceCode%2CReadME>

Visual Studio IDE Community 2017:

<https://www.visualstudio.com/ko/thank-you-downloading-visual-studio/?sku=Community&rel=15#>

컴퓨터 프로그래밍 언어인 C#을 활용해서 주식 자동매매 프로그램을 제작하는 방법을 담고 있습니다. 본 도서에서는 자동매매 프로그램을 제작할때 키움증권에서 제공하는 API를 사용하고 있습니다. 키움 증권 영웅문HTS에서 제공하는 기능들은 키움증권 API를 사용해서 프로그램으로 구현 할 수 있습니다. 로그인, 사용자정보 , 종목정보 , 장중 실시간데이터 처리 , 주문 , 조건식 검색 등의 모든 기능들은 프로그램에서 구현 가능합니다.

1장부터 9장까지 hts에서 제공하는 기능들을 구현하는 방법에 대해서 다룹니다. 10장에서는 1~9장에서 다룬 내용들을 바탕으로 자동매매 프로그램을 제작해봅니다.

1.3장에서는 자동매매 프로그램을 만들기 위한 기초적인 C# 프로그래밍 지식을 제공하고 있습니다. 하지만 글을 읽으면서 어려움을 느끼시거나 필요하시다면 C# 프로그래밍 언어를 따로 조금은 공부하시길 권합니다.

책에서는 각 장에서 만들고자 하는 기능을 구현하는 과정을 예제를 중심으로 살펴봅니다. API 를 사용하는 방법과 체크리스트 위주로 코드를 작성하여 이해를 돋고 있습니다. 체크리스트를 따라서 순서대로 코드를 작성하다보면 기능이 완성됩니다.

본 도서에서는 자동매매 프로그램을 만들면서 겪을 수 있는 어려움들과 해법을 제시합니다.

책을 다 읽으셨을때는 독자 여러분들 모두 하나의 자동매매 프로그램을 완성 할 수 있을 것입니다.

<제목 차례>

0. 시작하기 전에	5
0.1 주식을 프로그램으로 자동매매 하는 이유	6
0.2 자동매매 프로그래밍으로 할 수 있는 일	7
0.3 자동매매 프로그래밍 과정의 어려움	10
1. 자동매매 프로그래밍 시작하기	11
1.1 자동매매 프로그래밍 준비	12
1.2 키움증권 API	14
1.3 프로그램 화면 만들기	16
1.4 프로그래밍 언어 C#	27
02. 키움증권 API	40
2.1 API 사용 신청하기	41
2.2 API 설치하기 환경설정	46
2.3 KOA Studio 설치하기	54
3. 사용자 로그인	57
3.1 프로그램에서 HTS처럼 로그인하기	58
3.2 로그인 사용자 정보 확인	62
3.3 사용자 계좌 조회	72
4. 종목 검색	86
4.1 종목 조회	87
4.2 실시간 종목 조회	99
5. 호가창	110
5.1 호가창	111
6. 매수 주문	130
6.1 프로그램에서 매수 주문하기	131
6.2 매수금액 제한걸기	140
7. 매도 주문하기	156
7.1 사용자 주식 잔고 확인	157
7.2 주식 매도 주문 하기	170
8. 체결정보	184
8.1 OnReceiveChejanData 함수	185
9. 조건식	201
9.1 사용자 조건식 불러오기	202

9.2 조건식기반 종목 검색	207
9.3 실시간 조건식 편입 이탈 종목 검색	218
10. 조건식 기반 자동매매	235
10.1 프로그램 화면 구성 / 설명	236
10.2 종목 매수 주문	239
10.3 잔고 조회	243
10.4 주문 기록	255
10.5 종목 매도 주문	258
10.6 자동매매 거래규칙 설정	262
10.7 주문 정정 , 취소	267
10.8 자동매매 시작	274
10.9 자동매매 중지 , 전체 청산	284
부록 A : 기타 API 함수	286
차트 그리기	287
부록 B : 소스코드	295
Github	296

0. 시작하기 전에

00 장에서는 주식을 프로그램으로 자동매매 해야하는 이유에 대해서 이야기합니다. 그리고 앞으로의 글에서 다룰 자동매매 프로그램이 할 수 있는 놀랍고도 편리한 일들에 대해서 소개합니다. 자동매매 프로그래밍에서 겪을 수 있는 어려움과 어려움을 해결하기 위한 본 도서의 설명 방식을 다루고 있습니다.

이 책을 쓰는 이유

처음 자동매매 프로그램을 제작하려고 했을 때 여러가지 난관에 봉착했습니다. 예를 들어서 주가 데이터를 다운로드 하는 것인지, 다운로드를 받는다면 어떻게 다운로드 하는 것인지, 조건식은 프로그램에서 수식을 직접 적어서 구현해야 하는 것인지 등등 수많은 궁금증들이 있었습니다. 그래서 키움증권에서 제공하는 API 설명서를 읽어 봤습니다. 그리고 "KOA Studio"라는 API의 사용 설명과 코드 작성 예시가 담긴 프로그램도 살펴봤습니다. 하지만 KOA Studio 와 설명서를 살펴봐도 어떻게 코드를 작성하면 되는 것인지 쉽게 이해가 되지 않았습니다.

이 책은 제가 겪었던 이런 어려움을 겪고 있는 분들을 위해서 쓰여졌습니다. 프로그램에서 어떻게 주가 데이터를 요청하는지, API를 어떻게 사용하는지 쉽게 설명하고 있습니다. 자동매매 프로그램을 어떻게 만들 수 있는지 예제를 통해서 쉽게 따라 할 수 있도록 책의 내용을 서술했습니다.

이 책의 구성

- 1장. 자동매매 프로그램을 만들기 전에 필요한 사전지식을 소개합니다. 자동매매 프로그램을 만들기 위한 프로그래밍 언어 C#을 간략히 소개합니다.
- 2장. 자동매매 프로그래밍을 하기 위한 키움증권 API를 설치하는 방법을 다룹니다.
- 3장. 프로그램에서 hts처럼 로그인하고 사용자 정보를 조회해봅니다. 사용자 계좌 정보도 프로그램에서 확인해봅니다.
- 4장. 사용자의 프로그램에서 종목 정보를 조회하는 방법을 알아봅니다.
- 5장. 프로그램에서 호가창을 만들어봅니다.
- 6장. 프로그램에서 매수주문을 하는 방법을 알아보고, 매수 금액을 제한하는 방법을 배워봅니다.
- 7장. 사용자의 프로그램에서 주식 잔고를 확인하고, 매도주문을 요청해봅니다.
- 8장. 프로그램에서 실시간 주문 체결정보 확인해봅니다.
- 9장. 사용자의 조건식을 기반으로 프로그램에서 종목을 검색해봅니다. 실시간으로 조건식에 편입, 이탈되는 종목들도 조회해봅니다.
- 10장. 1~9장의 내용을 바탕으로 자동매매 프로그램을 만들어봅니다.

0.1 주식을 프로그램으로 자동매매 하는 이유

주식을 하는 사람들이 겪는 가장 큰 스트레스는 아마 하루종일 차트를 보고 있어야 한다는 점일 것입니다. 또한 차트를 보고 있어도 사고 파는 순간을 결정하기란 쉽지가 않습니다. 아무리 마음 속으로 정해놓은 규칙이 있다고해도 주문하는 순간이 되면 무수히 많은 갈등이 생깁니다. 차트를 보고 있지 않는 동안에도 머리 속은 온통 주식생각 뿐이라 해야할 일도 손에 잡히지 않습니다.

여러분이 주식에 전혀 신경쓰지 않아도 주식투자를 똑똑한 누군가가 알아서 대신해준다면 참 좋을 것입니다. 아니면 여러분의 투자 규칙에 맞춰서 주식을 대신 매도하고 매도 해 줄 사람이 있다면 참 편할 것입니다. 만약에 여러분이 정말 주식 투자를 대신해줄 누군가를 고용하기라도 한다면, 매 달 그 사람의 월급을 줘야 할 것입니다. 그리고 그 고용한 사람이 투자 실수라도 한다면 그 땐 큰 분쟁이 발생 할 수도 있습니다. 하지만 주식투자를 대신 해 줄 "누군가"가 로봇이라면 최소한 실수는 일어나지 않을 것입니다. "로봇"은 정해진 투자규칙대로 오차 없이 매수, 매도만 반복할 것입니다.

그렇다면 주식투자를 대신 해 줄 이런 "로봇" 같은 컴퓨터 프로그램을 가지고 있다면 참 편리할 것입니다. 만약에 그런 "로봇" 같은 프로그램이 있다면, 가격과 사용 방법이 먼저 궁금 할 것입니다. 하지만 여러분이 이런 자동매매 프로그램을 직접 만든다면 가격이 "무료"가 됩니다. 그런데 어떤 프로그램도 만들어 본 적이 없는데 자동매매 프로그램을 직접 만든다고 생각하면 그것은 굉장히 어려운 일처럼 느껴질 것입니다.

이 책에서 하고자 하는 것이 바로 로봇 같은 자동매매 프로그램을 만드는 방법을 소개하는 것입니다. 책에서는 주식 자동매매 프로그램을 만들기 위한 기초 프로그래밍 지식도 함께 제공하고 있습니다. 책의 프로그램 내용을 처음부터 끝까지 그대로 따라 적기만 하면 주식 자동매매를 할 수 있도록 책을 구성했습니다. 자세하게 캡처된 화면들을 보면 하나씩 따라 하시면 됩니다. 책을 다 읽으셨을때는 여러분의 자동매매 프로그램이 완성되어 있을것입니다.

0.2 자동매매 프로그래밍으로 할 수 있는 일

사람들은 흔히 일을 할 때 자신만의 원칙이 있습니다. 이것은 주식투자에 있어서도 예외는 아닙니다. 사람들은 주식 투자를 할 때, 각자 다양한 근거를 바탕으로 해당 종목을 매수하기로 결정합니다. 그리고 보유 종목을 매도 할 때도 각자 저마다의 이유를 바탕으로 종목을 매도합니다. 이렇게 주식 투자에 있어서 자신만이 가지고 있는 규칙을 증권사의 HTS(Home Trading System)에서는 하나의 수식으로 표현 할 수 있습니다.

그리고 HTS에서 표현되는 이러한 식을 "조건식"이라고 부릅니다. 주식을 투자자를 대신해서 자동으로 매매해주는 자동매매 프로그램에서는 "조건식"을 기반으로 종목을 매수하고 매도합니다.

본 도서로부터 자동매매 프로그래밍 지식을 충분히 습득한다면 다양하게 증권 관련 프로그램을 제작하는데 응용 할 수 있을 것입니다. 이 장에서는 본 도서를 통해서 만들게 될 자동매매 프로그램으로 할 수 있는 일들에 대해서 살펴보도록 하겠습니다.

- 자동매매 프로그램으로 할 수 있는 일
 - 조건식 기반 주가 모니터링
 - 투자종목 일괄 주문

자동매매 프로그램으로 할 수 있는 일

자동매매 프로그램은 크게 2가지의 일을 해낼 수 있습니다.

- 조건식 기반 편입 종목 모니터링
- 투자 종목 일괄 주문

자동매매 프로그램에서 할 수 있는 이 2가지 일에 대해서 살펴보겠습니다.

조건식 기반 주가 모니터링

투자자는 HTS에서 만든 조건식의 정보를 자동매매 프로그램으로 가져올 수 있습니다. 자동매매 프로그램에서 투자자는 실시간으로 자신의 조건식으로 검색되는 종목의 가격이 어떻게 변화하는지 알 수 있습니다.

The screenshot shows the KIWI Stock HTS interface with the following details:

- Search Criteria (조건식):** 대형 저평가 우량주
- Advanced Search (추천식):** 대상변경 > 업증대상(전체) > 제외없음 > 전체월결산
- Current Price Condition (현재가기준):**
 - 500: [+] 속역원 이상
 - 10: [-] 속역원 이상 50000: [=] 속역원 미하
- Screening Results (검색 결과):**

종목명	현재가	전일대비	등락률	거래량	시가	고가	저가
한국타이어월드	17,550	▼ 250	-1.40%	48,052	17,750	17,900	17,500
SK하이닉스	87,000	▲ 500	+0.58%	3,411,673	88,500	88,800	86,900
DB하이텍	13,350	▼ 50	-0.37%	178,758	13,550	13,750	13,350
넥센타이어	12,200	▲ 100	+0.83%	44,726	12,150	12,250	12,100
태광산업	1,320,000	▲ 13,000	+0.99%	588	1,289,000	1,329,000	1,289,000
쌍용양회	24,950	▼ 350	-1.36%	861,722	25,300	27,300	24,600
LG	80,200	▼ 500	-0.62%	153,720	81,000	81,700	79,800
현대제철	53,900	▼ 700	-1.28%	213,240	54,400	54,800	53,600
POSCO	352,500	▼ 6,000	-1.67%	132,872	358,500	360,000	350,500
삼성전자	2,645,000	▲ 38,000	+1.46%	359,272	2,669,000	2,682,000	2,622,000
- Buttons at the bottom:**
 - 분류별 (radio button)
 - 기나다순 (radio button)
 - 관심저장 (button)
 - 조건만족종목 (button)
 - 검색결과 (button)
 - 50 (button)
 - 100%진행 (button)
 - 비교분석 (button)
 - 성과검증 (button)

[키움증권 영웅문HTS 조건검색화면]

프로그램 사용자는 가져온 조건식들을 바탕으로 프로그램에서 종목들을 검색 할 수도 있습니다.

The screenshot shows the KIWI Stock HTS interface with the following details:

- Search Criteria (조건식):** 대형 저평가 우량주
- Advanced Search (추천식):** 대상변경 > 업증대상(전체) > 제외없음 > 전체월결산
- Current Price Condition (현재가기준):**
 - 500: [+] 속역원 미상
 - 10: [-] 속역원 미상 50000: [=] 속역원 미하
- Screening Results (검색 결과):**

종목명	현재가	전일대비	등락률	거래량	시가	고가	저가
한국타이어월드	17,550	▼ 250	-1.40%	48,052	17,750	17,900	17,500
SK하이닉스	87,000	▲ 500	+0.58%	3,411,673	88,500	88,800	86,900
DB하이텍	13,350	▼ 50	-0.37%	178,758	13,550	13,750	13,350
넥센타이어	12,200	▲ 100	+0.83%	44,726	12,150	12,250	12,100
태광산업	1,320,000	▲ 13,000	+0.99%	588	1,289,000	1,329,000	1,289,000
쌍용양회	24,950	▼ 350	-1.36%	861,722	25,300	27,300	24,600
LG	80,200	▼ 500	-0.62%	153,720	81,000	81,700	79,800
현대제철	53,900	▼ 700	-1.28%	213,240	54,400	54,800	53,600
POSCO	352,500	▼ 6,000	-1.67%	132,872	358,500	360,000	350,500
삼성전자	2,645,000	▲ 38,000	+1.46%	359,272	2,669,000	2,682,000	2,622,000
- Buttons at the bottom:**
 - 분류별 (radio button)
 - 기나다순 (radio button)
 - 관심저장 (button)
 - 조건만족종목 (button)
 - 검색결과 (button)
 - 50 (button)
 - 100%진행 (button)
 - 비교분석 (button)
 - 성과검증 (button)

투자종목 일괄주문

그런데 만약 이렇게 조건식으로 검색된 종목이 100개인데 하나씩 모두 찾아서 매수 주문을 한다면, 매수 주문을 모두 하는데 시간이 오래 걸리게 됩니다.

첫 번째로 종목을 매수 할때는 검색 한 시점의 가격으로 매수 주문을 할 수 있습니다. 하지만 조건검색으로 검색된 100번째 종목을 매수주문 할때는 가격이 많이 바뀌어 있을 수도 있습니다.

자동매매를 하면 프로그램이 조건검색으로 검색된 100개의 종목을 한번에 모두 매수주문 할 수 있습니다.



매도주문을 할때도 똑같이 한번에 매도 주문을 할 수 있습니다.

이렇게 하면 매수,매도 주문을 하는 타이밍을 놓치지 않고 주문을 원하는 시점에 바로 주식을 주문할 수 있습니다.

자동매매 프로그램은 실시간으로 조건검색으로 검색 되는 종목을 알아낼 수도 있습니다.

그리고 실시간으로 조건검색 화면에서 검색되는 종목을 바로 매수 할 수도 있습니다.

이렇게 프로그램으로 주식 거래를 진행하면 매수,매도 타이밍을 놓치지 않고 주문을 요청 할 수 있습니다.

0.3 자동매매 프로그래밍 과정의 어려움

0.3장에서는 자동매매 프로그래밍 과정에서 겪을 수 있는 어려움과 그 어려움을 극복 할 수 있도록 필자가 설명하는 방식을 소개하고 있습니다.

차근차근 따라하며 만드는 프로그램

예제와 체크리스트를 통해서 지루하지 않고 쉽게 따라 할 수 있도록 글을 적었습니다.

책의 설명 예시

이번 장에서는 종목의 이름을 검색하고, 종목을 주문해보도록 하겠습니다. 이번 장에서는 아래와 같은 순서로 코드를 작성하도록 하겠습니다.

- 로그인
- 화면에 사용자 정보 출력
- 사용자 조건식으로 종목 검색
- 종목 매수 주문

API를 사용해서 사용자 정보를 요청하는 방법.....

화면에 사용자 정보 출력하는 코드.....

한가지 기능의 설명이 끝날 때마다 체크리스트가 체크됩니다. 프로그램을 한 부분씩 차근차근 만들어보면서 학습 할 수 있도록 설명했습니다. API를 사용하는 방법과 예제 위주로 코드를 작성하여 이해를 돋고 있습니다.

자동매매 프로그램을 만들면서 겪을 수 있는 어려움

자동매매 프로그램을 만들면서 여러분은 여러가지 어려움을 겪을 수 있습니다. KOA Studio와 설명서를 다운로드 받았는데 어떻게 사용해야 하는 것인지 설명이 이해가 되지 않을 수 있습니다. 프로그래밍을 전혀 해본 적이 없어서 자동매매 프로그램을 만드는 것이 어렵게만 느껴질 수도 있습니다. 만드는 과정이 너무 오래 걸리진 않을까 걱정이 앞설 수도 있습니다. 본 도서에서는 기능 구현을 위한 소스코드를 모두 제공하고 있습니다. 프로그래밍을 전혀 모르는 독자들도 프로그램을 만들 수 있도록 1.3장에서는 자동매매 프로그래밍을 위한 간략한 프로그래밍 지식도 제공하고 있습니다.

1. 자동매매 프로그래밍 시작하기

Preview

1장에서는 자동매매 프로그램을 만들기 전에 준비해야 할 것들을 소개합니다. 자동매매 프로그램을 만들기 위해서는 증권사의 API, 프로그래밍에 대한 지식이 필요합니다. 하나씩 학습하며 자동매매 프로그램을 만들기 위한 기초를 다지도록 합니다.

Contents

- 1.1장에서는 프로그래밍과 API에 대해서 간략히 소개합니다.
- 1.2장에서는 키움증권 API를 사용하는 이유, API의 사용 설명서인 KOA Studio에 대해서 알아봅니다.
- 1.3장에서는 프로그램의 화면을 만드는 방법을 배웁니다.
- 1.4장에서는 프로그래밍 언어 C#으로 코드를 작성하는 방법을 알아봅니다.

1.1 자동매매 프로그래밍 준비

이 장에서는 자동매매 프로그래밍을 하기 전에 알아야 할 개념들을 소개합니다. 여러분은 키움증권에서 제공하는 API를 사용해서 자동매매 프로그램을 만들게 되실 것입니다. 그렇다면 자동매매 프로그래밍을 하기 전에 먼저 프로그램이 무엇인지, 키움증권에서 제공하는 API라는 것이 무엇인지 먼저 알아야 합니다.

프로그램

컴퓨터는 사람에 명령에 따라 일을 합니다. 사람들은 컴퓨터의 계산기 프로그램을 사용해서 산술 계산을 하기도 하고, 비디오 플레이어를 사용해서 영화를 보기도 합니다. 이처럼 컴퓨터에게 어떠한 명령을 내릴 때 사람들은 프로그램을 사용합니다. 프로그램은 사용자가 컴퓨터에게 보내는 명령어들의 묶음입니다. 계산기 프로그램에는 덧셈 명령, 뺄셈 명령, 곱셈 명령 등의 산술 연산 명령어들이 있습니다. 비디오 플레이어 프로그램에는 영상 재생, 일시 정지 등의 명령이 들어있습니다. 본 도서에서 다루는 자동매매 프로그램에는 주식 매수 주문, 매도 주문 등의 명령어들이 있을 것입니다.

프로그래밍 언어

컴퓨터는 사람의 말을 알지 못합니다. 트랜지스터 회로 속의 켜짐, 꺼짐 신호를 통해서 정보를 구분합니다. 컴퓨터에게 보내는 명령어를 작성 할 때는 사람의 언어가 아닌 컴퓨터에게 명령을 할 수 있는 언어로 명령어를 작성합니다. 이렇게 컴퓨터에게 명령을 하는 언어를 프로그래밍 언어라고 부릅니다. 본 도서에서는 C#이라는 프로그래밍 언어를 사용할 것입니다.

API

키움증권에서 제공하는 API라는 것이 무엇인지 알아보도록 하겠습니다. 키움증권 영웅문 hts에서는 많은 기능들을 제공합니다. 영웅문 hts에서는 조건식 기반 종목검색, 종목정보 조회, 차트 데이터 등의 정보를 알아 볼 수 있습니다. 하지만 이 많은 기능들을 모두 자주 사용하지는 않습니다. 그리고 사용자들은 몇가지 기능들을 조합해서 자신만의 투자 규칙을 만들어보기도 합니다.

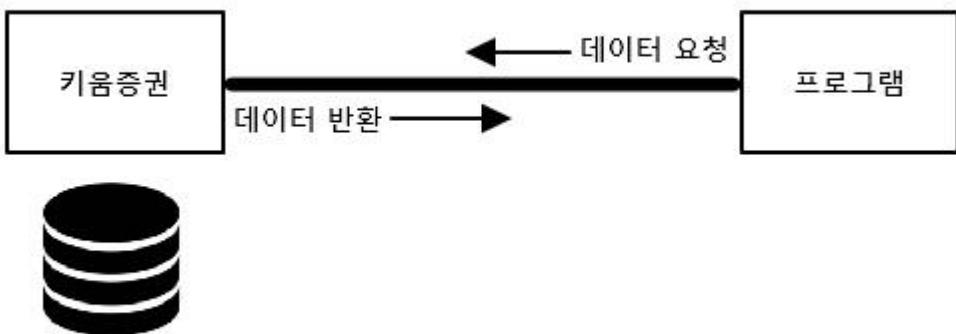
자신만의 투자규칙을 접목한 자동매매 프로그램을 만들기 위해서는 몇가지 정보가 필요합니다. 예를 들어서 종목을 정해진 규칙대로 매수, 매도하는 프로그램에서는 정해진 매수 규칙, 주식 가격 정보, 매도 규칙이 필요합니다. 주식 가격 정보는 키움 증권에서 가지고 있는 주가 데이터입니다.

만약에 키움 증권의 서버에 접근해서 데이터를 누구나 쉽게 다운로드 받고 접근 할 수 있다면 자동매매 프로그램을 만들때 API라는 것이 필요 없을 것입니다. 하지만 증권사의 데이

터를 이렇게 누구나 접근 할 수 있게 공개한다면 보안상 위험합니다. 그렇다면 키움 증권의 중요한 데이터는 보호하면서 개발자들이 프로그래밍은 쉽게 할 수 있는 뭔가가 필요해집니다. 키움증권에서는 API를 제공해서 이런 필요를 충족하고 있습니다.

만약 키움 증권 API를 사용해서 만들어진 프로그램이 많아지게 된다면 점점 더 많은 투자자들이 프로그램을 사용하기 위해서 키움증권에서 계좌를 개설 할 것입니다. 안정적으로 데이터를 제공해서 프로그램을 만들 수 있게 하고, 더 많은 사용자를 확보하기 위해서 키움증권에서는 API를 제공합니다.

API는 쉽게 설명하면 사용자의 프로그램과 키움증권 서버를 이어주는 다리입니다.



건축에 빗대어 설명하면 사용자가 프로그램이라는 건축물을 짓기 위해서는 시멘트를 가져와야 합니다. 시멘트는 건축을 하기 위한 재료입니다. 키움증권이 가지고 있는 데이터가 시멘트입니다. 시멘트를 가지려 가기 위한 길목에 있는 다리가 바로 API입니다. API를 지나서 데이터를 가져오고 프로그램을 만들게 됩니다. 사용자는 이 API라는 경로로 데이터를 요청하고 자동매매 프로그램을 만들 수 있습니다.

1.2 키움증권 API

앞으로 사용할 키움증권에서 제공하는 API외에도 여러 증권사에서 API를 제공하고 있습니다. 대신증권, 한국투자증권에서도 API를 제공합니다.

이 장에서는 증권사에서 제공하는 API들 중에서 키움증권 API를 사용하는 이유에 대해서 이야기 해보겠습니다.

키움증권 API를 사용하는 이유

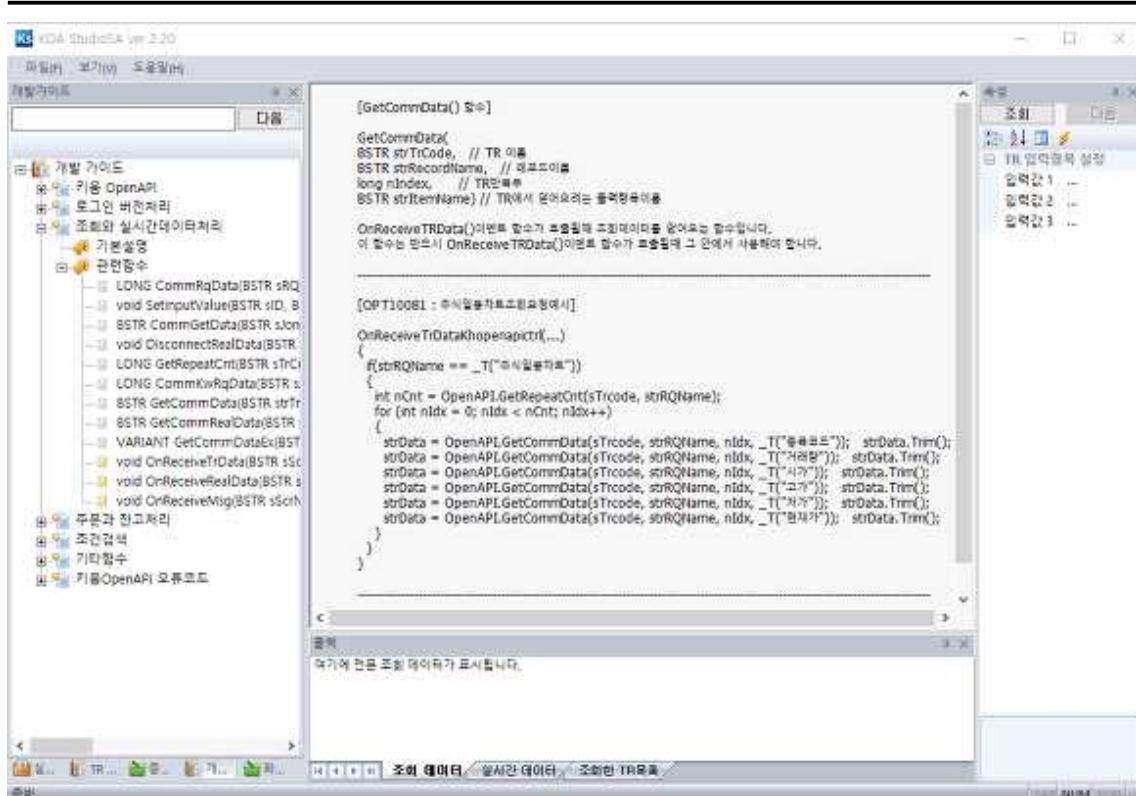
키움증권 API를 사용하는 이유는 크게 2가지입니다.

- 가장 많은 HTS 사용자
- KOA Studio의 풍부한 예제

가장 많은 HTS 사용자

현재 주식거래를 하는 투자자들이 가장 많이 사용하고 있는 HTS가 키움증권의 영웅문 HTS 일 것이라고 생각합니다. 자동매매 프로그램은 기본적으로 사용중인 HTS의 화면 속 기능들을 그대로 구현 할 수 있습니다. 앞으로 사용자가 가장 많은 키움증권 영웅문 HTS의 기능을 구현하는 자동매매 프로그램을 만들어보도록 하겠습니다.

KOA Studio의 풍부한 예제



[키움증권 KOA Studio 화면]

키움증권에서는 키움증권 API의 사용 설명서인 "KOA Studio" 프로그램을 제공합니다. KOA Studio 에는 개발자가 어떻게 프로그램에서 키움증권에 데이터를 요청하고, 받을 수 있는지 명시되어 있습니다. 설치하는 방법은 2.3장에서 알아봅니다.

1.3 프로그램 화면 만들기

1.3장에서는 자동매매 프로그래밍을 시작하기 전에 C# 프로그래밍에 대해서 간략히 알아보도록 하겠습니다. C# 프로그래밍 언어에 관한 더 많은 내용이 있지만 이번 장에서는 이 책에서 직접적으로 쓰이는 아래 내용들에 대해서만 다룹니다. 1.3장에서는 프로그램의 화면을 만드는 방법에 대해서 배워봅니다.

- Visual Studio 설치하기
- 화면 설명
- 화면에 버튼 가져다 놓기

Visual Studio 설치하기

본 도서에서는 프로그래밍 언어 C#을 사용해서 자동매매 프로그램을 만듭니다. C# 같은 프로그래밍 언어로 기술한 명령어가 모여있는 글을 소스코드라고 부릅니다. 프로그램을 만들기 위해서는 소스코드를 어딘가에 모아서 잘 작성해야 합니다. 메모를 편리하게 하기 위해서 "메모장"이라는 프로그램이 있습니다. C# 프로그램을 작성하기 위해서는 Visual Studio라는 프로그램을 사용하면 편리합니다.

아래 과정을 따라하면 Visual Studio를 설치 할 수 있습니다.

먼저 구글에 접속합니다.



Visual Studio 를 검색합니다.

The screenshot shows the Google search results for the query "visual studio". The top result is a link to the Visual Studio IDE page, which includes a large orange arrow pointing to it. Below this are links for Visual Studio 2017, Visual Studio Community 2017, and Visual Studio 2017 15.6. On the right side of the search results, there is a sidebar for Microsoft, featuring the Microsoft logo, a search bar, and several links to Microsoft products like Office, TFS, and Sharepoint.

검색결과 약 112,900,000건 (0.36초)

Visual Studio IDE, Code Editor, VSTS, & App Center
https://www.visualstudio.com/ ▶ 이 페이지 번역하기
Visual Studio dev tools & services make app development easy for any platform & language. Try our Mac & Windows code editor, IDE, or VSTS for free.
Visual Studio Downloads: Free Visual Studio | Visual Studio IDE | Pricing

Downloads | IDE, Code, & Team Foundation Server | Visual Studio
https://www.visualstudio.com/downloads/ ▶ 이 페이지 번역하기
5월 전 - Download Visual Studio Community, Professional, and Enterprise. Try Visual Studio Code or Team Foundation Server for free today.

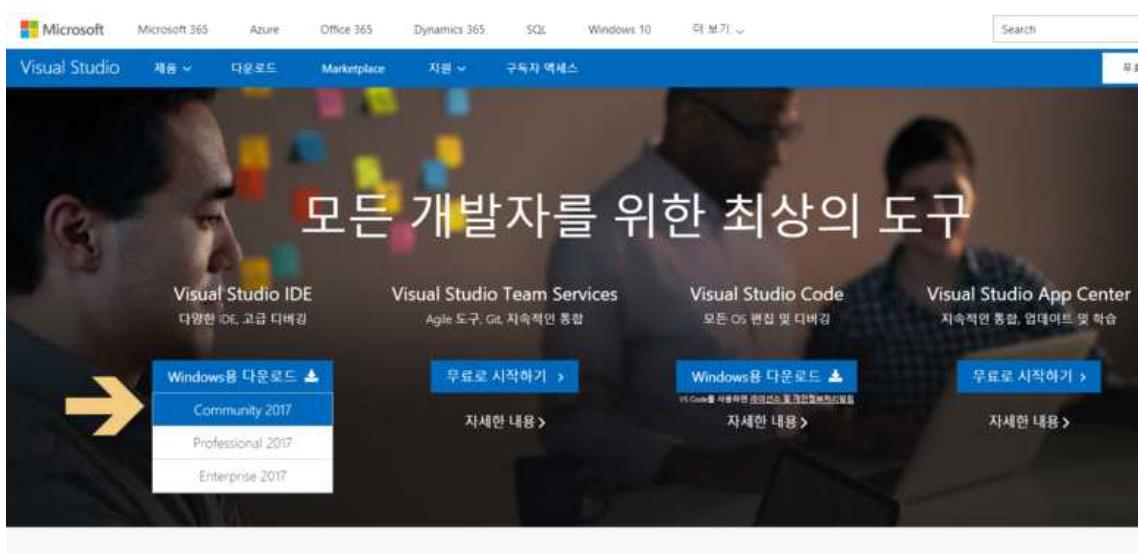
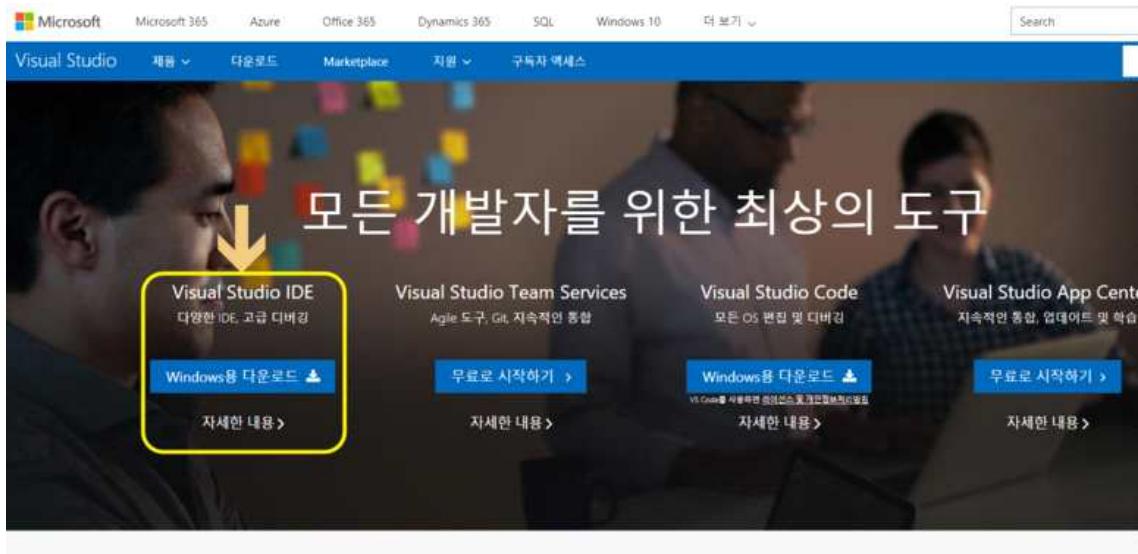
Visual Studio 2017 설치 | Microsoft Docs
https://docs.microsoft.com/ko-kr/visualstudio/install/install-visual-studio ▶
2017.12.4. Visual Studio를 설치하는 방법을 단계별로 알아봅니다.
1단계 - Visual Studio에 대해 ... 3단계 - Visual Studio 설치 ... 4단계 - 워크로드 선택

다운로드 Visual Studio Community 2017 | Imagine
https://imagine.microsoft.com/ko-kr/Catalog/Product/530 ▶
코드 편집, IntelliSense, 리팩터링, 코드 수정, 디버깅을 강화하게 때문에 언어 및 플랫폼과 상관없이 빠르고 간편하게 월등한 작업을 할 수 있습니다. DevOps를 적용하는 팀의 경우 Visual Studio 2017을 사용하면 Live Unit Testing, 놀라운 마이크로세스 총력형 투명성 검사와 같은 새로운 개념의 실시간 기능으로 코드 훑기를 빠르게 전개하고...

Visual Studio 설명서 | Microsoft Docs
https://docs.microsoft.com/ko-kr/visualstudio/ ▶
Visual Studio 2017을 사용하여 플랫폼 및 디자인에 대해 선택된 언어로 응용 프로그램, 서비스 및 도구를 개발하는 방법에 대해 알아보는다.

Visual Studio 2017 15.6 릴리스 정보 | Microsoft Docs
https://docs.microsoft.com/ko-kr/visualstudio/releasenotes/vs2017-relnotes ▶
2018.4.4 - Visual Studio 2017의 향상된 최고 기능에 대한 릴리스 정보입니다. Visual Studio를 사용하여 대 규모으로 개발하고, 함께 협업하거나 품질을 제공하세요.

첫번째 검색 결과를 클릭합니다.

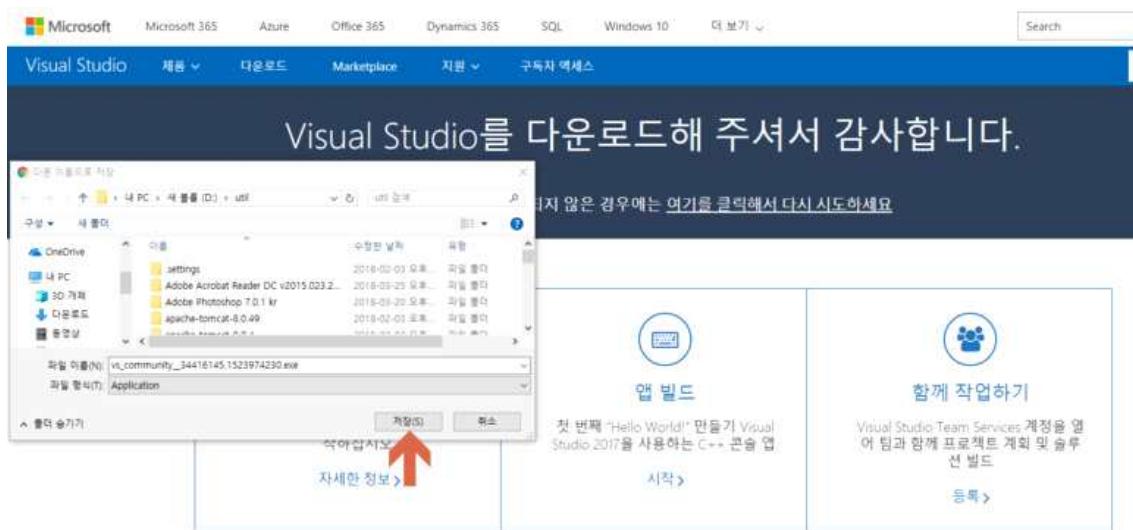


Community 2017을 선택합니다.

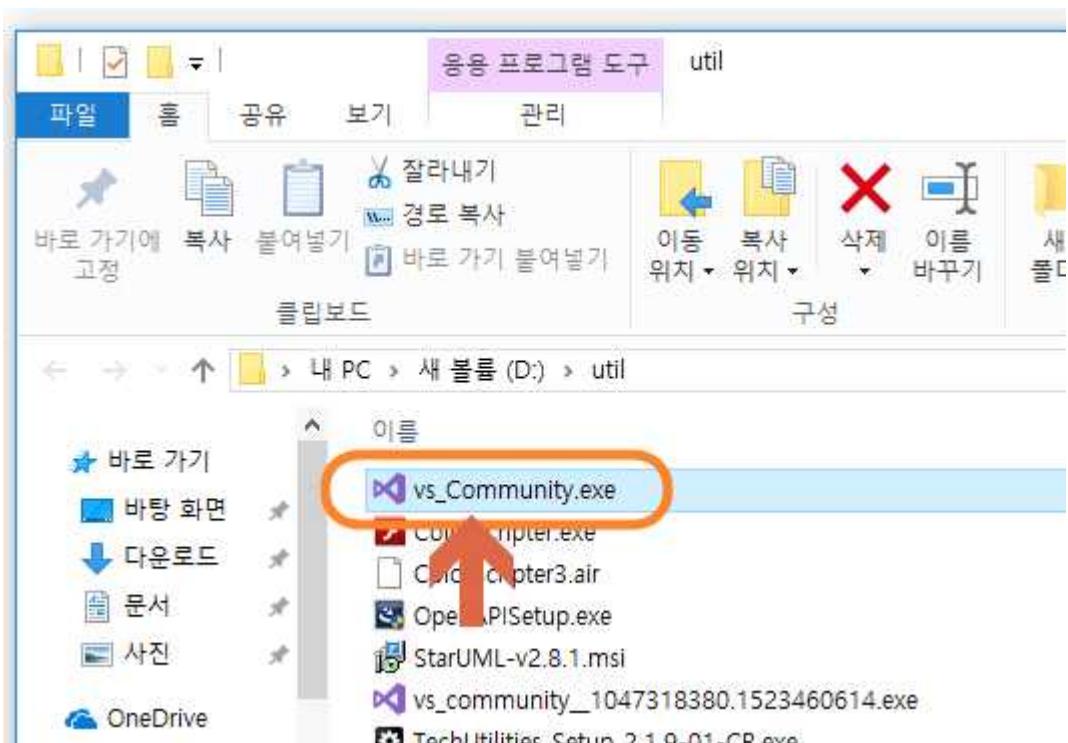
아래 링크를 클릭하시면 바로 다운로드 페이지로 이동 하실 수 있습니다.

※다운로드 페이지 링크

<https://www.visualstudio.com/ko/thank-you-downloading-visual-studio/?sku=Community&rel=15#>



링크에 들어가면 다운로드가 진행됩니다. 저장을 클릭하고 다운로드를 진행해주세요.



다운로드 한 파일을 실행합니다.

×

Visual Studio Installer

시작하기 전에 설치를 구성할 수 있도록 몇 가지 항목을 설정해야 합니다.

개인정보처리방침에 대해 자세히 알아보려면
[Microsoft 개인정보처리방침](#)을 참조하세요.

계속하면
Microsoft 소프트웨어 사용 조건에 동의하는 것입니다.

계속(C)

계속을 클릭하고 진행합니다.

Visual Studio Installer

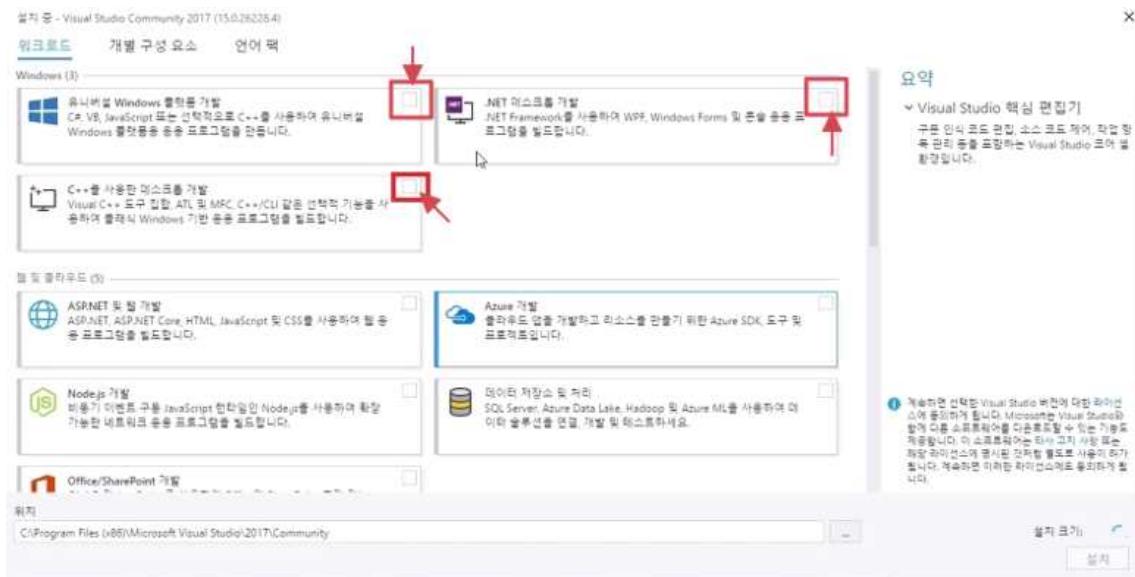
거의 완료되었습니다... 모든 항목을 준비하는 중입니다.

✓ 다운로드됨

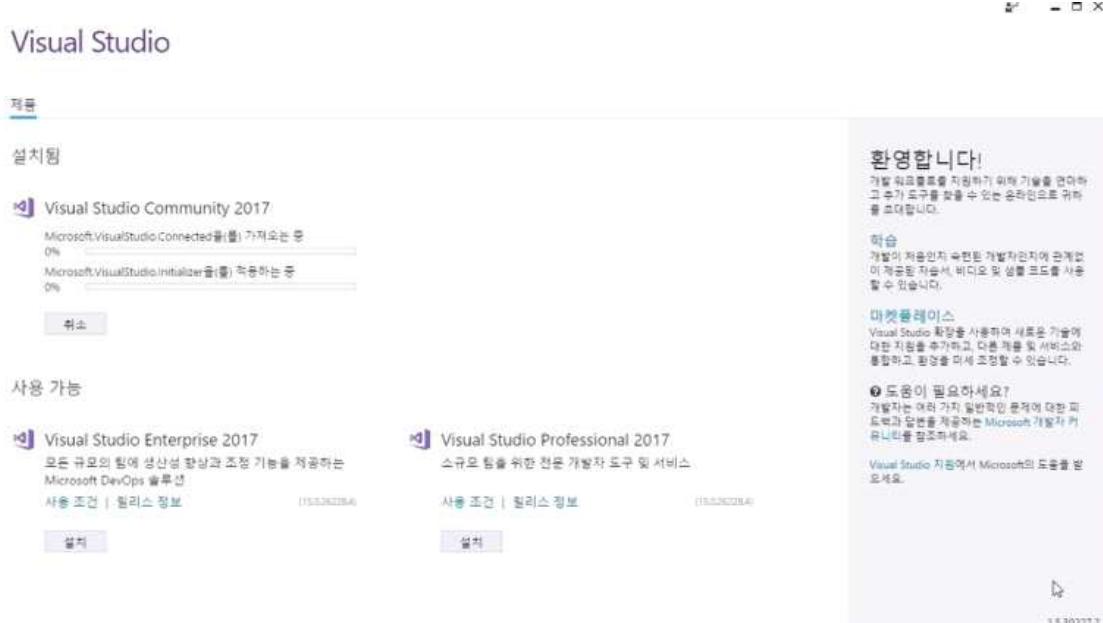


✓ 설치됨

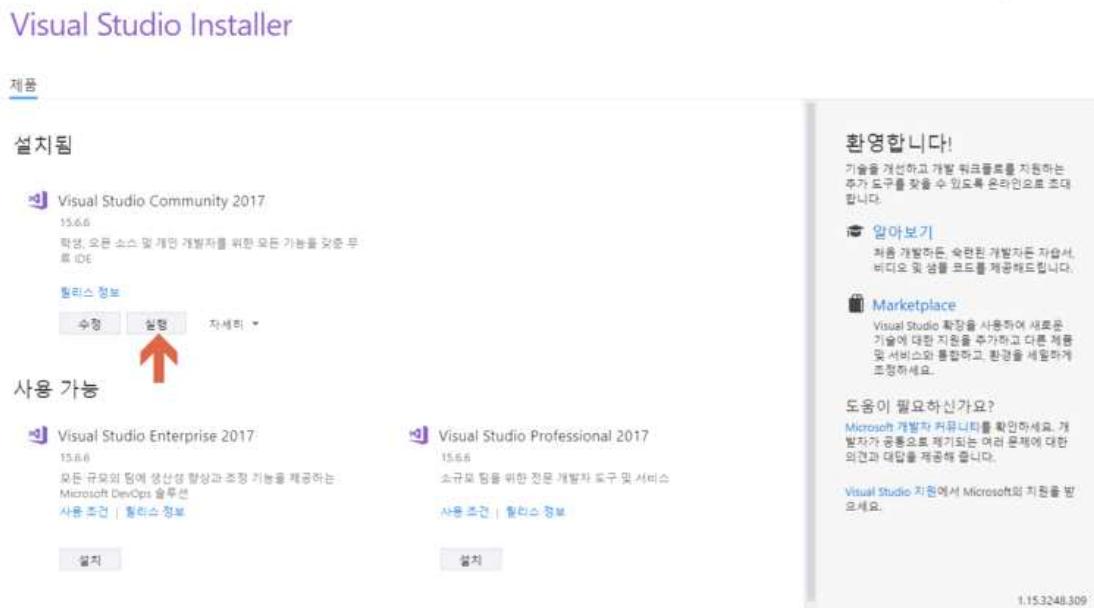




3가지 항목을 체크하고 오른쪽 아래의 설치 버튼을 클릭합니다.



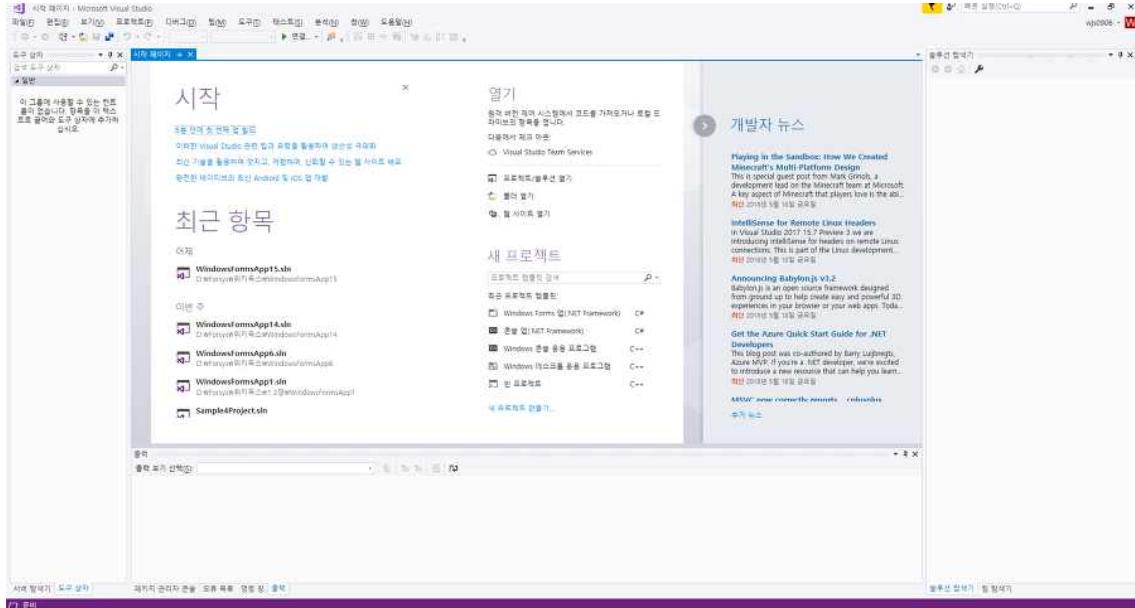
설치가 진행되고 있는 모습입니다.



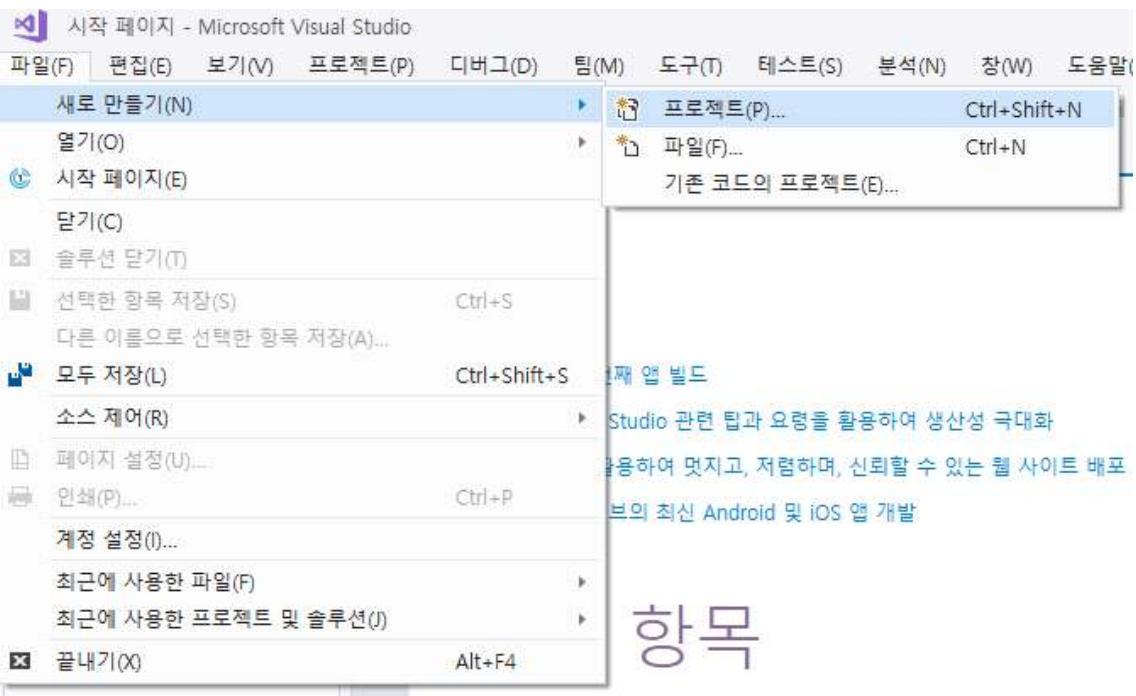
실행 버튼을 클릭하면 Visual Studio 가 실행됩니다.

화면 설명

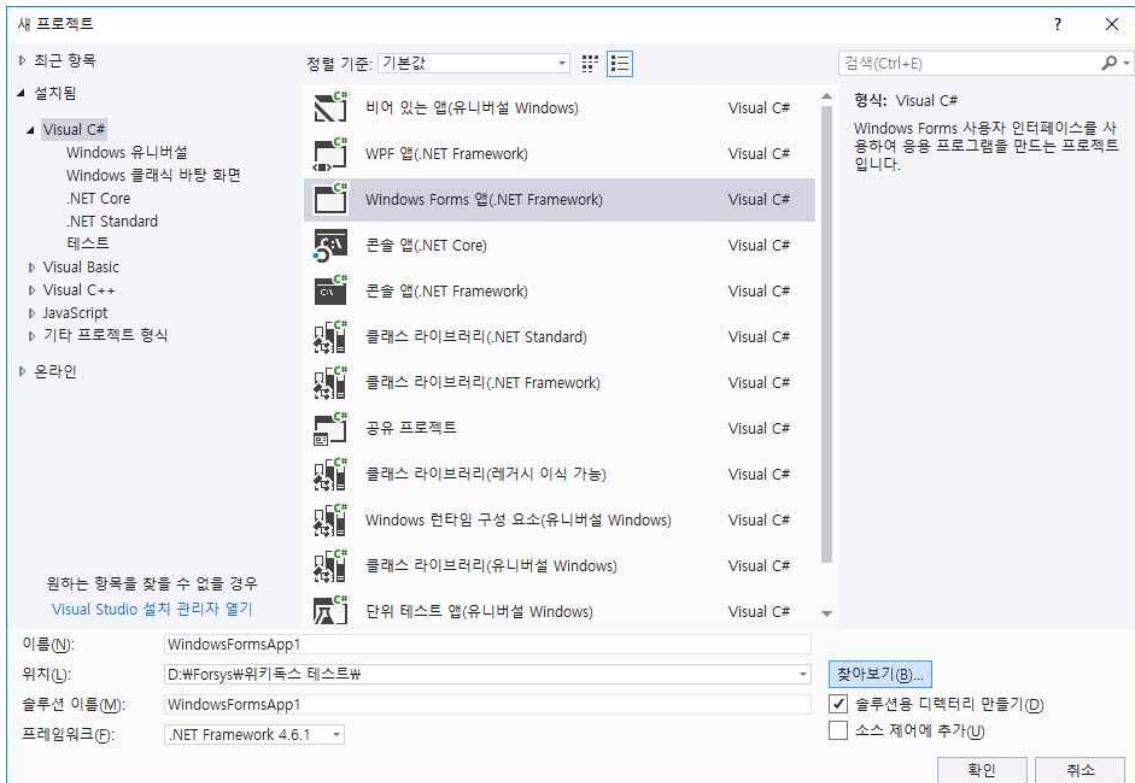
이제 앞으로 사용하게 될 Visual Studio의 환경을 구경해보도록 하겠습니다. 소스코드를 어디에 어떻게 작성하게 되는지 궁금 할 것입니다. 아래 그림은 Visual Studio의 실행 화면입니다.



위쪽에 파일, 편집, 보기, 프로젝트, 디버그, 팀, 도구, 테스트, 분석, 창, 도움말 순서대로 탭이 있는 것이 보입니다. 많은 버튼들도 보입니다. Visual Studio의 어디에 어떻게 코드를 작성하는지 알아보기 위해서 한번 코드를 작성해보도록 하겠습니다.

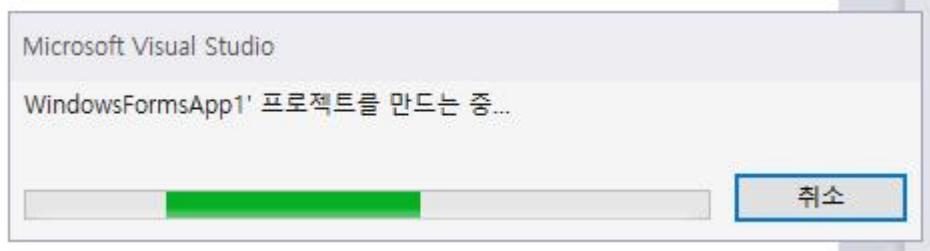


먼저 파일(F) – 새로만들기(N) – 프로젝트(P) 을 클릭합니다. 새로운 프로젝트를 만드는 화면이 나타납니다. 보통 회사에서 프로젝트는 하나 하나의 커다란 작업 단위입니다. 우리는 지금 자동매매 프로그램을 만든다는 하나의 프로젝트를 진행하고 있습니다. Visual Studio에서의 프로젝트는 일상 생활에서의 프로젝트와 같은 의미입니다.

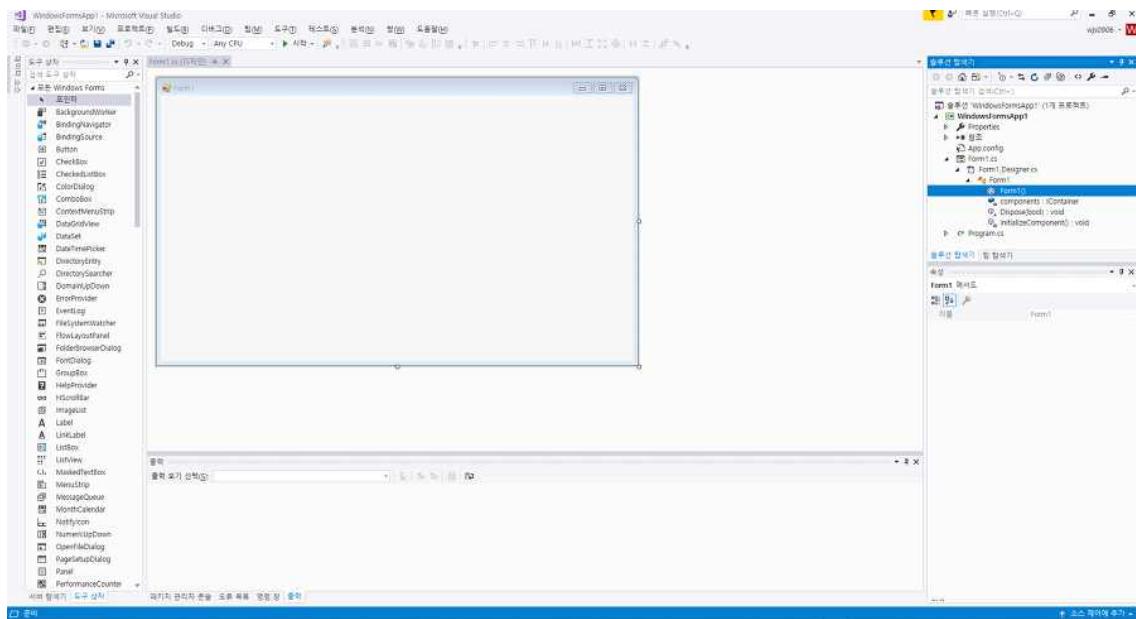


Windows Forms 앱을 선택합니다. 프로젝트의 이름과 저장할 위치를 선택하고 확인을 클릭

합니다.



잠시 기다리면 프로젝트가 생성됩니다.



프로젝트가 생성된 화면입니다. 지금 보고 있는 화면은 Form1.cs[디자인] 화면입니다. 왼쪽의 수많은 아이콘들이 있습니다. 이 아이콘들을 드래그 해서 화면을 구성하게 됩니다. "컨트롤"이라고 부릅니다. 오른쪽에는 프로젝트 안의 파일들이 보입니다. 앞으로 Form1()이라는 파일에서 소스코드를 작성하게 될 것입니다.

소스코드를 작성하는 Form1() 파일도 살펴보도록 하겠습니다.

The screenshot shows the Microsoft Visual Studio interface. The top window displays the code for `Form1.cs` (디자인), which contains the standard boilerplate code for a Windows Form application. Below it, the Windows Forms Designer shows a blank white form window. The bottom part of the interface includes the Solution Explorer, Properties, and Task List panes.

```

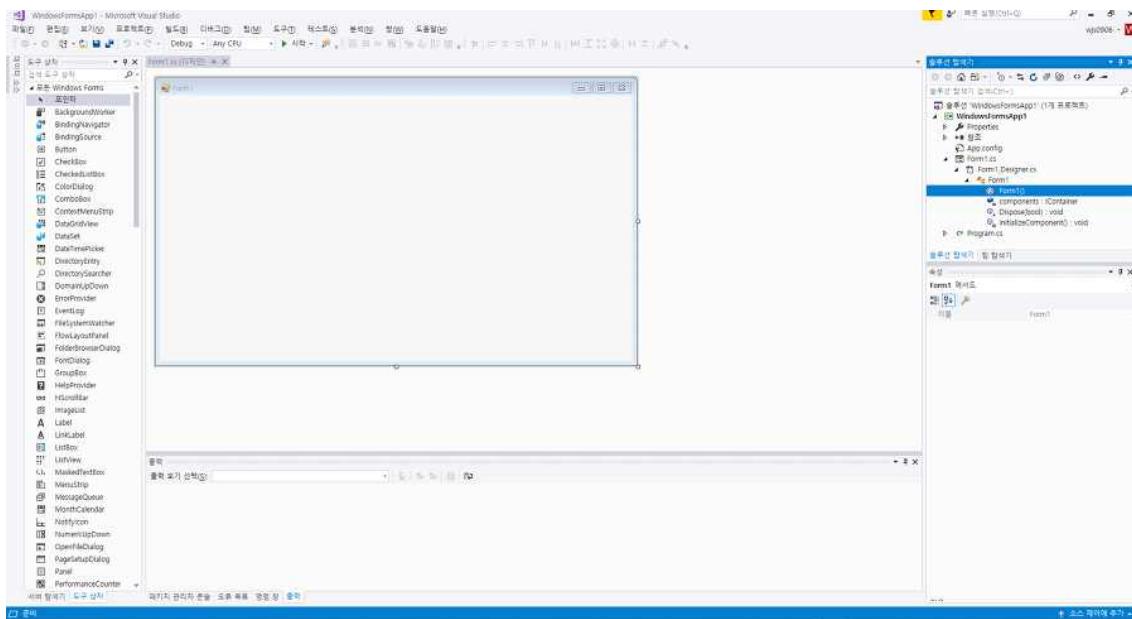
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace WindowsFormsApp1
12 {
13     public partial class Form1 : Form
14     {
15         public Form1()
16         {
17             InitializeComponent();
18         }
19     }
20 }
21

```

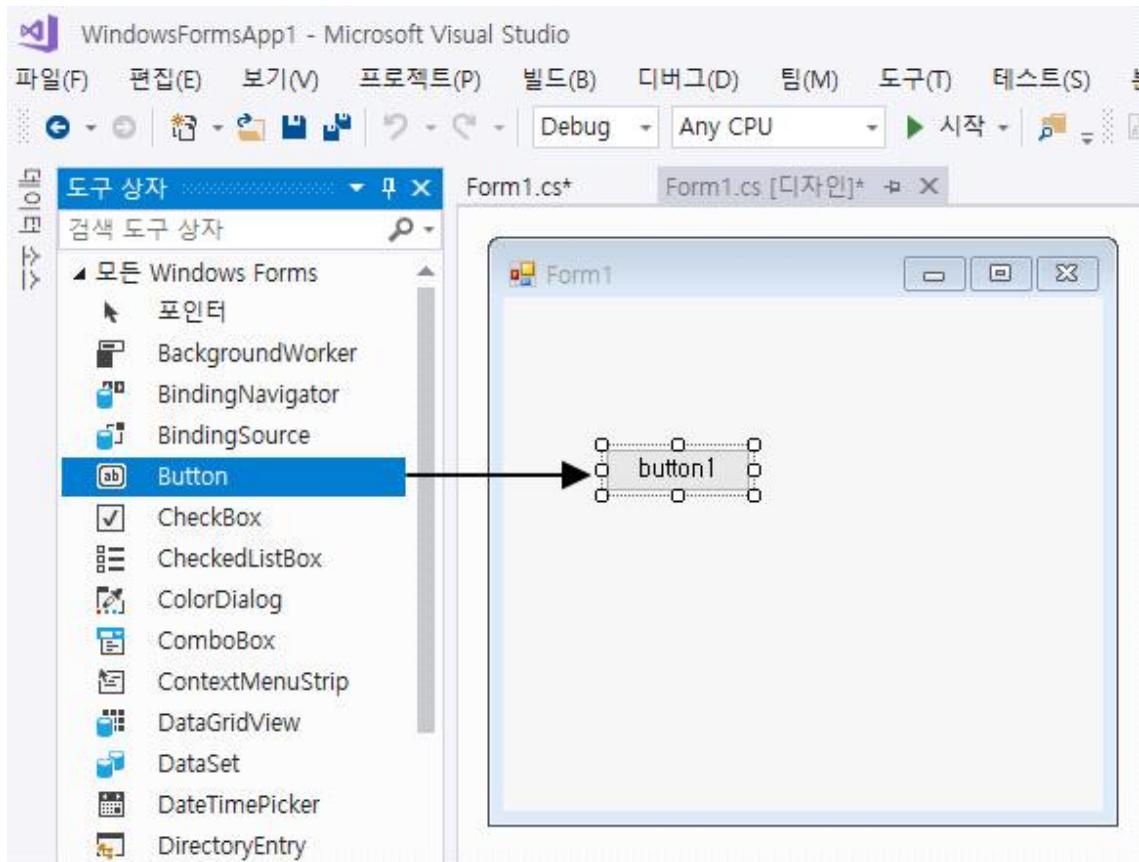
Form1.cs 파일입니다. cs는 C Sharp 의 줄임말입니다. 앞으로 이 파일에 소스코드를 작성하게 됩니다.

화면에 버튼 가져다놓기

화면에 버튼을 한번 배치해보도록 하겠습니다. 화면에 버튼을 배치하기 위해서 `Form1.cs` [디자인] 화면을 클릭합니다.



화면의 왼쪽 컨트롤 목록으로부터 버튼 컨트롤을 드래그해서 가져옵니다.



화면에 버튼이 생성되었습니다.

1.4 프로그래밍 언어 C#

1.4장에서는 자동매매 프로그래밍을 시작하기 전에 C# 프로그래밍에 대해서 간략히 알아보도록 하겠습니다. C# 프로그램ming 언어에 관한 더 많은 내용이 있지만 이번 장에서는 이 책에서 직접적으로 쓰이는 아래 내용들에 대해서만 다룹니다. 1.4장에서는 프로그램의 코드를 작성할 때 알아야 할 내용을 다룹니다.

- 변수
 - 연산자
 - 조건문
 - 반복문
 - 함수
 - 클래스
-

변수

시스템 트레이딩 프로그램에서 주식을 자동으로 익절, 손절 하려면 주가 정보를 수집해서 몇 % 오르고 내렸는지 알아야 합니다. 이 때 컴퓨터 프로그램에서는 주식 가격의 정보를 알고 있어야 합니다. 그리고 현재 가격의 정보도 가지고 있으면서 처음 구매했던 가격과 비교해야 합니다.

이렇게 자동매매를 할 때 프로그램에서 어떠한 정보를 저장하기 위해 컴퓨터의 메모리에 할당하는 공간을 **변수**라고 합니다.

그러면 앞으로 만들게 될 프로그램에서 변수를 어떻게 사용하는지 알아보도록 하겠습니다. 프로그램에서 변수를 사용하려면 메모리의 공간을 사용하겠다는 "선언"을 소스코드에 적어야 합니다. 변수는 아래와 같은 형태로 선언합니다.

자료형 변수이름;

변수이름은 메모리에 할당하는 공간인 "변수"의 이름입니다. 자료형은 변수에 담을 수 있는 값의 형태입니다. 변수에는 "문자열", 정수, 실수, true, false 등의 값을 저장 할 수 있습니다.

자료형	표현	범위
int	정수	-2,147,483,648 ~ 2,147,483,647
long	정수	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
float	소수	소수점 이하 7자리까지 표현
double	소수	소수점 이하 15자리까지 표현
string	문자열	문자열
bool	참, 거짓	true, false

소스코드에서 어떻게 변수를 선언하는지 살펴보도록 하겠습니다. 변수는 아래와 같이 선언할 수 있습니다.

```
public void variable_Example()
{
    int integer; // -2,147,483,648 ~ 2,147,483,647
    long big_integer; // -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

    float rational_number; // 소수점 이하 7자리까지 표현
    double big_rational_number; // 소수점 이하 15자리까지 표현

    string letters; // 문자열
    bool flag; // true, false
}
```

변수는 메모리에 할당된 공간이라고 앞서 설명했습니다. 메모리 공간에는 값을 집어 넣을 수 있습니다. 이것을 "변수에 값을 대입한다." 라고 합니다.

=을 사용해서 변수에 값을 대입 할 수 있습니다.

```
public void variable_Example()
{
    int integer = 5;
    long big_integer = 10000;

    float rational_number = 1.234;
    double big_rational_number = 1.23456;

    string letters = "시스템트레이딩";
    bool flag = true;
}
```

이렇게 변수에 초기 값을 대입하는 것을 변수의 "초기화" 라고 부릅니다.

연산자

시스템 트레이딩 프로그램에서 주식의 현재 가격이 매수 가격보다 몇 % 올랐는지 판단하기 위해서는 산술적인 연산이 필요합니다. 예를 들어서 1000원에 매수했던 종목이 20% 오른 가격은 얼마인지 계산하려면 아래와 같이 계산 할 수 있습니다.

$$1000 + (1000 * 0.2) = 1200$$

이런 산술적인 연산에 쓰이는 +, -, *, / 등의 연산자를 프로그램의 소스코드에서 사용 할 수 있습니다.

프로그램에서 어떻게 연산자를 사용하는지 아래 예시를 통해서 보도록 하겠습니다.

```
public void operator_Example()
{
    int number1 = 20;
    int number2 = 7;
    int number3;

    number3 = number1 + number2;//number3의 값 : 27
    number3 = number1 - number2;//number3의 값 : 13
    number3 = number1 * number2;//number3의 값 : 140
    number3 = number1 / number2;//number3의 값 : 2
}
```

변수 number1, number2에 저장한 값을 산술 연산하고 number3에 저장 할 수 있습니다. 여기서 한가지 주목할 부분이 있습니다. number1 / number2의 값이 2로 number3에 저장되고 있습니다. 원래 $20 / 7$ 을 하면 $2.8571\dots$ 가 나옵니다. $2.8571\dots$ 이 아닌 2가 number3에 저장되는 이유는 변수 number3의 타입이 int형이라서 그렇습니다. 소수 값을 int형 변수에 저장하면 소수점 뒤의 자리가 버립니다.

산술 연산자는 변수의 값을 하나씩 증가 시킬수도 있고, 감소 시킬수도 있습니다.

```
public void operator_Example()
{
    int number = 20;

    number++; // number 1증가
    number--; // number 1감소
}
```

C#에서는 이렇게 산술 연산자를 사용 할 수 있습니다. 아래 코드처럼 변수를 더하면서 대입 할 수도 있습니다.

```
public void operator_Example()
{
    int sum = 0;
    sum += 1;
}
```

`sum += 1`은 `sum = sum + 1` 과 같습니다. 이렇게 sum에 `sum+1`을 대입하게 되면 0이었던 변수 sum의 값이 1을 더한 값으로 덮어쓰기가 됩니다.

그리고 C#에서는 항목을 비교하는 비교 연산자도 사용 할 수 있습니다. 예를 들어서 주식의 현재 가격이 매수한 가격보다 20% 오른 것이 맞는지 판단하기 위해서는 매수시점의 가격과 현재 가격을 비교해야 합니다.

2가지의 항목을 비교하는 비교 연산자의 종류는 6가지가 있습니다.

1. ==
2. !=
3. >
4. <
5. >=
6. <=

코드와 함께 연산자를 알아보겠습니다.

```
int buyPrice = 1000;
int currentPrice = 1200;

buyPrice == currentPrice://buyPrice와 currentPrice 가 같은가?
buyPrice != currentPrice://buyPrice와 currentPrice 가 다른가?
buyPrice > currentPrice://buyPrice가 currentPrice보다 큰가?
buyPrice < currentPrice://buyPrice가 currentPrice보다 작은가?
buyPrice >= currentPrice://buyPrice가 currentPrice보다 크거나 같은가?
buyPrice <= currentPrice://buyPrice가 currentPrice보다 작거나 같은가?
```

각각의 연산자가 가지는 의미를 같이 적었습니다. ==는 두가지 항목이 같은지를 비교합니다. !=는 두가지 항목이 다른지를 비교합니다. >, <, >= , <= 부등호는 두가지 항목의 크기를 비교합니다. 비교 연산자는 비교가 끝나고 나면 결과로 true , false 값을 가집니다.

예를 들어서 buyPrice 와 currentPrice의 값이 같으면 buyPrice == currentPrice의 값은 true가 됩니다. buyPrice 와 currentPrice의 값이 다르다면 buyPrice == currentPrice의 값은 false가 됩니다. 앞서 변수에서 bool 타입의 변수에 true 또는 false 값을 저장 할 수 있었습니다. 아래 코드처럼 bool 타입의 변수에 비교 연산자의 결과를 저장하는 것이 가능합니다.

```
public void operator_Example()
{
    int buyPrice = 1000;
    int currentPrice = 1200;
    bool result;

    result = buyPrice == currentPrice://buyPrice와 currentPrice 가 같은가?
    result = buyPrice != currentPrice://buyPrice와 currentPrice 가 다른가?
    result = buyPrice > currentPrice://buyPrice가 currentPrice보다 큰가?
    result = buyPrice < currentPrice://buyPrice가 currentPrice보다 작은가?
    result = buyPrice >= currentPrice://buyPrice가 currentPrice보다 크거나 같은가?
    result = buyPrice <= currentPrice://buyPrice가 currentPrice보다 작거나 같은가?
}
```

변수 `result`에 비교 결과를 저장하고 있습니다. 비교 연산자의 값이 `true` 또는 `false`를 갖는다는 사실을 잘 기억하고 조건문으로 넘어가도록 하겠습니다.

조건문

시스템 트레이딩 프로그램에서는 주식의 현재 가격이 일정 %보다 올랐을 때, 손절 % 보다 내렸을 때 매도 주문을 요청합니다. 특정 종목의 가격이 오르고 내림에 따라서 일정, 손절로 동작을 구분해야 합니다. 프로그램에서 어떻게 조건문을 사용해서 여러 가지 상태를 구분하는지 예시를 통해서 알아보도록 하겠습니다.

영어에서 `if`는 만약에 라는 의미를 가지고 있습니다. 문장으로 작성한다면 아래와 같은 형태일 것입니다.

if 주식의 가격이 20% 올랐다면, 매도주문을 요청

프로그램의 소스코드에서도 비슷하게 작용합니다. 프로그램의 소스코드에서는 아래와 같은 형태로 구문이 작성됩니다. 아래 예시는 형태이며 코드를 정말로 이렇게 작성하지는 않습니다.

```
if(주식의 가격이 20% 올랐다면){
    매도 주문을 요청
}
```

`if`문의 괄호 안에 있는 "주식의 가격이 20% 올랐다면" 문장을 연산자에서 배운 비교 연산자로 바꿔보도록 하겠습니다.

```
if ( 현재가격 == ( 매수가격 + ( 매수가격 * 0.2 ) ) )
{
    매도 주문 요청
}
```

가격이 20% 올랐다면 (`매수가격 + (매수가격 * 0.2)`)의 값이 현재 가격과 같을 것입니다. 앞의 연산자에서 비교연산자는 `true` 또는 `false`의 값을 가진다고 했습니다.

`if`문은 괄호 안의 값이 아래 코드처럼 `true`이면 {} 중괄호 안의 문장을 실행합니다. 반대로 `false`이면 중괄호 안의 문장을 실행하지 않습니다.

```
if(true)
{
    문장 실행
}
```

```
if(false)
{
    문장 실행하지 않음
}
```

그러면 아래와 같이 코드를 작성한다면 if() 괄호 안에는 비교 연산자의 결과로 true , false 값이 들어갑니다. true, false 값에 따라서 {} 중괄호 안의 내용을 실행합니다.

```
if ( 현재가격 == ( 매수가격 + ( 매수가격 * 0.2 ) ) )
{
    매도 주문 요청
}
```

정리하면 소스코드 if문의 괄호 안에 걸려내고 싶은 조건을 적습니다. 그리고 {} 중괄호 안에 조건에 해당할때 동작할 행동을 적습니다.

이번에는 올바르게 if 문을 사용해서 소스코드를 적은 예시를 보겠습니다.

```
public void if_Example()
{
    int buyPrice = 1000;
    int currentPrice = 1200;

    if (currentPrice == (buyPrice + (buyPrice * 0.2 ) ) )
    {
        //매도주문 요청
    }
}
```

위 코드처럼 코드를 작성한다면 종목이 올랐을때 매도 주문을 요청 할 수 있을 것입니다. 그런데 주식을 거래 할때는 손절을 해야 하는 경우도 있습니다. 두가지 이상의 조건을 함께 작성하려면 else , else if 를 사용합니다. else 문은 if 문에 해당하지 않는 모든 경우를 의미합니다. else if 문은 if 문의 조건에는 해당하지 않지만 따로 처리해야 하는 다른 조건이 있을때 사용합니다. 아래 예시를 통해서 else 문부터 살펴보도록 하겠습니다.

```
public void if_Example()
{
    int currentPrice = 1200;

    if (currentPrice == 1000 )
    {
        //현재 가격이 1000원인 경우
    }
    else
    {
        //현재 가격이 1000원이 아닌 모든 경우
    }
}
```

앞쪽의 if문에서는 currentPrice 가 1000과 같은지 비교하고 있습니다. else 문은 ()괄호가 없고 {} 중괄호만 있습니다. 이렇게 코드를 작성하면 else문에서는 currentPrice가 1000이 아닌 모든 경우를 처리하게 됩니다. 이제 else if 문을 살펴보도록 하겠습니다.

```
public void if_Example()
{
    int buyPrice = 1000;
    int currentPrice = 1200;

    if (currentPrice == (buyPrice + (buyPrice * 0.2) ) ) )
    {
        //익절 주문 요청
    }
    else if (currentPrice == (buyPrice - (buyPrice * 0.2) ) ) )
    {
        //손절 주문 요청
    }
    else
    {
        //주식 보유
    }
}
```

else if문의 () 괄호 안에는 if() 문의 조건에 해당하지 않지만 else 문으로부터 독립되어 따로 처리 될 조건이 작성됩니다.

반복문

시스템 트레이딩 프로그램에서는 매수 조건식에 해당하는 종목이 여러 개일 경우 여러 개의 종목을 모두 같은 방식으로 매수합니다. 이 때 같은 형태의 매수주문이 여러번 요청되게 됩니다. 이렇게 반복적인 동작을 수행 할 때는 **반복문**을 사용해서 소스코드를 작성하면 편리 합니다. 예를 들어서 조건식으로 검색되는 종목이 종목1부터 종목 50까지 매수주문을 요청하는 상황을 가정해보겠습니다. "종목 1부터 50까지" 는 반복을 수행해야하는 특정 조건에 해당합니다. "매수 주문"은 반복적으로 수행해야하는 동작에 해당합니다. 이것을 글로 적으면 아래와 같이 적을 수 있습니다.

```
조건식으로 검색되는 종목1부터 종목50까지{
    매수주문 요청
}
```

한글로 작성된 문장 순서를 코드로 살펴보도록 하겠습니다. 아래 예시는 반복문 for문을 사용해서 작성되었습니다.

```
for(int i = 1 ; i <= 50 ; i++)
{
    //종목 i 매수
}
```

소스코드에서 반복문은 예시처럼 for문이라고 불리는 문장을 사용해서 작성합니다.

for문의 구조를 살펴보도록 하겠습니다. for문의 괄호 안에는 아래 3가지 정보를 담고 있습니다.

1. 반복의 시작지점
1. 반복의 종결시점
1. 반복 횟수

각각의 정보는 for문에서 아래 코드와 같이 위치합니다.

```
for(반복의 시작지점; 반복의 종결시점 : 반복횟수)
{
    반복할 문장
}
```

반복의 시작시점은 변수로 선언합니다. 아래 코드를 보면 int형 변수 i의 값이 1로 초기화되고 있습니다. 반복의 종결 시점은 $i \leq 50$ 이라고 작성되어 있습니다. i의 값이 50이 될 때까지 {} 중괄호 안의 문장을 반복 수행함을 의미합니다. 반복횟수는 i라고 작성되어 있습니다. i는 i가 1씩 증가함을 의미합니다.

```
for(int i = 1 ; i <= 50 ; i++)
{
    //종목 i 매수
}
```

정리하면 i의 값이 1부터 50까지 1씩 증가 할 것이고, i의 값이 50이 될 때까지 {} 중괄호 안의 문장을 반복실행하라

이런 뜻이 됩니다.

앞의 예시들처럼 반복문을 작성하면 아래와 같습니다.

```
public void repeat_Example()
{
    for (int i = 0; i <= 50; i++)
    {
        //종목i 매수
    }
}
```

함수

앞으로 만들게 될 자동매매 프로그램에선 2가지 종류의 함수를 다루게 됩니다. 1가지는 일반함수이고 나머지 1가지는 이벤트 함수입니다. 먼저 일반함수에 대해서 살펴보도록 하겠습니다.

시스템 트레이딩 프로그램에서는 반복적으로 수행되는 일련의 동작들이 많습니다. 예를 들어서 사용자가 프로그램에서 주가를 조회하고 가격이 20,000원 이상인 종목을 반복적으로 모두 매수하려고 하는 상황을 가정해보겠습니다. 사용자는 먼저 주식의 가격을 조회합니다. 그리고 종목의 매수를 진행합니다. 이 과정은 항상 같은 방식으로 수행해야 합니다. 프로그램에서는 이런 정해진 일련의 동작들을 하나의 함수로 묶어서 정의하고 사용 할 수 있습니다.

한글로 작성된 코드 예시를 보겠습니다.

```
주식 종목 조회, 매수 함수(){
    //1. 주식 종목의 가격을 조회한다.
    //2. 종목의 가격이 20,000원 이상이면 매수한다.
}
```

소스코드로 작성한 모습을 보겠습니다.

```
public void method_Example()
{
    //1. 종목의 가격 조회
    //2. 종목의 가격의 20,000원 이상이면 매수
}
```

소스코드에서 함수의 형태를 조금 더 자세히 살펴보도록 하겠습니다. 함수는 원래 아래와 같은 형태로 작성합니다.

```
public 리턴타입 함수의 이름( 입력자료형1 입력변수1, 입력자료형2 입력변수2)
{
    return 리턴값;
}
```

프로그램에서 사용하는 함수와 수학에서 배운 함수는 본질적으로 같은 의미입니다. 수학에서는 $f(x)$ 의 x 에 특정 값을 대입하면 y 라는 결과 값이 도출됩니다.

프로그램에서의 함수도 입력 변수 자리에 입력 변수를 대입하면 {}괄호 안에서 결과 값을 도출하고 return 문장을 통해서 결과 값을 돌려줍니다.

덧셈을 하는 함수를 만든다면 아래와 같이 코드를 작성 할 수 있습니다.

```
public int plus_Example(int a,int b)
{
    return a + b;
}
```

예제 함수에서는 int형 변수 a와 b를 입력 받고 a+b를 리턴값으로 반환하고 있습니다. a+b는 int형 값이기 때문에 함수의 "리턴타입" 자리에 int를 적어줍니다.

이렇게 반복적인 코드를 함수로 작성했다면 사용 할 수 있어야 합니다. 함수를 사용하는 것을 프로그램에서는 "호출"한다고 합니다. 함수는 아래와 같이 호출합니다.

```
public void method_test()
{
    method_Example(); //함수의 이름을 적어서 호출합니다.
}
public void method_Example()
{
    //1. 종목의 가격 조회
    //2. 종목의 가격의 20,000원 이상이면 매수
}
```

위의 코드는 아래 코드와 같은 동작의 결과가 같습니다.

```
public void method_test()
{
    //1. 종목의 가격 조회
    //2. 종목의 가격의 20,000원 이상이면 매수
}
```

이렇게 함수로 코드를 작성하고 호출하는 것은 코드의 길이가 길어질수록 진가를 발휘합니다. 함수를 사용하지 않고 예제의 코드를 작성한다면 예제에서는 1,2 두가지 동작의 코드를 작성하면 됩니다. 하지만 만약에 실행해야 하는 문장이 20줄, 50줄 이상이 된다면 매번 20줄, 50줄의 코드를 작성해야 합니다. 이렇게 함수로 작성해두면 20줄, 50줄의 코드를 한번만 작성하고 소스코드의 다른 곳에서 호출해서 편리하게 사용 할 수 있습니다.

이제 이벤트 함수에 대해서 살펴보도록 하겠습니다. 이벤트는 화면에서 일어나는 모든 일을 지칭하는 말입니다. 이벤트 함수는 화면에서 이벤트가 발생했을 때 동작하는 함수입니다. 이벤트 함수는 아래와 같은 형태입니다.

```
public void method_test()
{
    //1. 종목의 가격 조회
    //2. 종목의 가격의 20,000원 이상이면 매수
}
```

이벤트 함수의 입력변수는 object 타입의 변수와 EventArgs 타입의 변수로 구성되어 있습니다. object 타입의 변수 sender는 이벤트가 어디서 발생했는지를 담고 있습니다. EventArgs 타입의 변수 e에는 이벤트의 정보가 담겨 있습니다. 아래 예제를 통해서 더 자세히 살펴보도록 하겠습니다.

만약에 화면에서 버튼을 클릭했을때 동작하는 이벤트 함수를 작성한다면 아래와 같이 코드를 작성하면 됩니다.

```
public Form1()
{
    InitializeComponent();
    button1.Click += buttonClicked;
}
```

button1은 화면 개체의 이름입니다. button1 뒤에 있는 .Click 은 button1을 클릭하는 이벤트를 의미합니다. += 은 덧셈과 대입 연산을 의미합니다. buttonClicked 는 함수의 이름입니다. 코드에서는 buttonClicked라는 함수를 button1.Click 에 더해서 대입하고 있습니다. 이렇게 대입하면 button1을 클릭했을때 buttonClicked 함수가 동작하게 됩니다.

하지만 아직 buttonClicked 함수가 어떤 동작을 수행하는지는 정의되지 않았습니다. 아래 코드처럼 Form1()에 함수를 추가하고 함수의 내용을 아래에 작성하면 함수의 동작을 정의하고 동작하게 할 수 있습니다.

```
public Form1()
{
    InitializeComponent();
    button1.Click += buttonClicked;
}

public void buttonClicked(object sender, EventArgs e)
{
    if(sender.Equals(button1)){ //이벤트를 발생시킨 객체가 button1인지 확인
        int number1 = 0;
        int number2 = 10;
        int number3 = number1 + number2;
    }
}
```

클래스

시스템 트레이딩 프로그램에서는 사용자의 주식 잔고를 조회하거나 조건식을 조회 할 수 있습니다. 그 외에도 다른 여러가지 정보들을 조회 할 수 있습니다. 그리고 1가지 정보를 조회 할때 조회하는 정보 안에 포함된 여러가지 정보들이 있습니다. 아래 예시를 보겠습니다.

주식 종목 정보를 조회하는 경우

사용자는 종목 정보를 조회하는 경우에 종목의 현재가격 , 거래량 , 시가, 고가 , 저가 , 등락율 등의 정보들을 알 수 있습니다.

이런 여러가지 정보들을 하나의 루트으로 관리하면 정보를 관리하기 편리합니다. 프로그램에서는 실세계에 존재하는 이런 하나 하나의 개체들을 모델링해서 루트으로 관리 할 수 있습니다.

한글로 작성한 모습을 먼저 보겠습니다. 아래와 같은 모습으로 종목의 정보가 루트일 수 있습니다.

한글로 작성한 모습을 먼저 보겠습니다. 아래와 같은 모습으로 종목의 정보가 루트일 수 있습니다.

```
종목{
    종목코드
    종목명
    현재가
    등락율
    거래량
}
```

클래스는 아래와 같은 형태로 선언 할 수 있습니다.

```
class 클래스이름
{
```

아래 예제를 통해서 더 자세히 살펴보도록 하겠습니다. 아래 예제는 주식 종목을 클래스로 모델링 했습니다.

```
class Stock
{
    string stockCode;//종목코드
    string stockName;//종목명
    int stockPrice;//현재가
    float changeRate;//등락율
    long volume;//거래량
}
```

주식 종목 말고 다른 개체를 모델링 한 예제도 살펴보겠습니다. 프로그램에서는 실세계에 존재하는 하나의 객체를 class를 사용해서 모델링합니다. 실세계에는 종목처럼 특정 동작을 수행하지 않는 객체도 있지만 어떤 "행동"을 할 수 있는 객체도 있습니다.

예를 들어서 자동으로 매매하는 거래 규칙을 모델링 해보겠습니다. 여기서 거래 규칙은 아래 4가지 변수를 갖습니다.

- 거래규칙 고유번호
- 거래규칙이 종목을 매수할때 사용하는 조건식 이름
- 거래규칙의 이익 %
- 거래규칙의 손실 %

거래 규칙에 거래 규칙만의 고유한 매도 규칙도 있다면 아래와 같이 클래스를 작성 할 수 있습니다.

```
class AutoTradingRule
{
    int autoTradingRuleNumber;//거래 규칙 고유번호
    string purchaseConditionName;//거래규칙이 종목을 매수할때 사용하는 조건식 이름
    float profitRate;//거래규칙의 이익 %
    float lossRate;//거래규칙의 손실 %

    public void autoTradingSell(int currentPrice)//거래 규칙의 매도 규칙 (객체의 행동)
    {
        if (currentPrice == (currentPrice+(currentPrice *0.2)))
        {
            //매도주문
        }
        else if (currentPrice == (currentPrice - (currentPrice * 0.2)))
        {
            //매도주문
        }
    }
}
```

클래스는 예제처럼 변수와 메소드(함수)로 구성되어 있습니다.

02. 키움증권 API

Preview

2장에서는 자동매매 프로그램을 만들기 위한 환경을 구축합니다. 차례대로 따라하며 앞으로 자동매매 프로그램을 만들기 위한 환경을 잘 구성해보도록 합니다.

Contents

2장에서는 키움증권 API에 대해서 알아봅니다.

2.1장에서는 키움증권 홈페이지에서 키움증권 API 사용을 신청합니다.

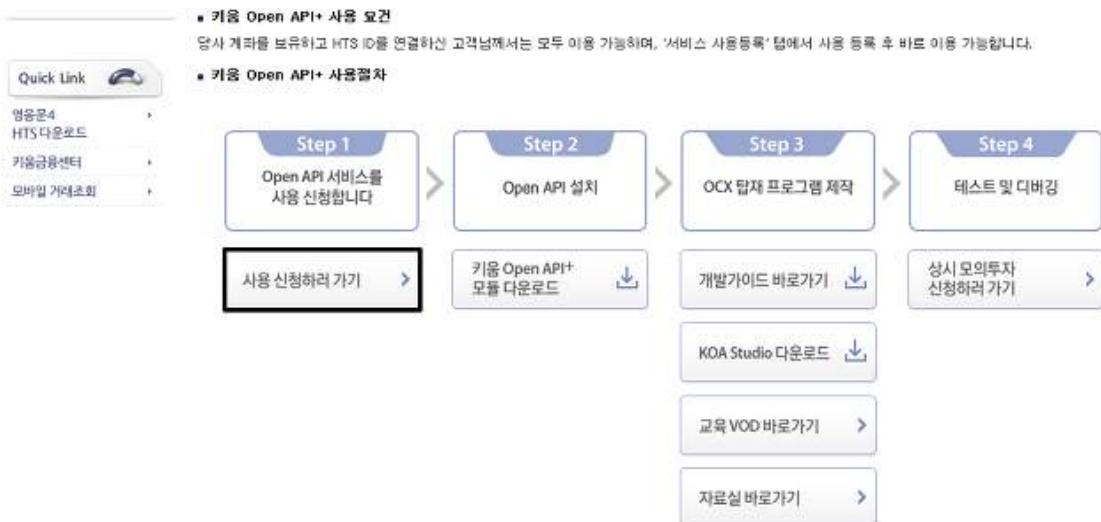
2.2장에서는 API를 컴퓨터에 설치하고, 프로그래밍을 위한 환경을 설정합니다.

2.3장에서는 키움 API 사용 설명서인 KOA Studio를 설치합니다.

2.1 API 사용 신청하기

2.1장에서는 키움증권 API를 사용하기 위해서 어떻게 사용 신청을 할 수 있는지 알아보도록 하겠습니다.

키움 API 사용신청



■ STEP 1. Open API 사용신청 합니다.

-키움 Open API+ 서비스 이용을 위하여는 사용 등록 신청이 필요합니다.

-당사 '홈페이지 > 트레이딩 채널 > Open API > 키움 Open API+ > 서비스 사용 등록/해지' 탭에서 사용등록이 가능합니다.

■ STEP 2. OpenAPI+ 모듈 다운로드 및 설치

-사용 신청 후 키움 Open API+ 모듈을 다운로드 하여 설치합니다.

■ STEP 3. Open API OCX 탑재 프로그램 제작 및 KOA Studio 활용

-키움 Open API+는 프로그램 개발 확장성을 고려하여 OCX 컨트롤로 제작 지원합니다.

※ 사용자 편의에 따라 VB, 액셀, Access, MFC 등으로 프로그램 제작이 가능합니다.

-데이터 요청 및 수신은 TR 서비스명을 검색한 후 OCX를 통해 API 함수를 호출하는 방식으로 진행됩니다.

프로그램 제작 편의를 위해 개발가이드 및 KOA Studio를 제공하고 있습니다. 가이드 및 자료실에 게시 된 샘플, 교육 VOD를 활용하시면 개발 시 도움을 받으실 수 있습니다.

※ KOA Studio는?

-당사가 제공하는 TR의 목록과 정보를 간편하게 확인 할 수 있습니다.

-TR을 테스트할 수 있는 화면을 제공하며, 입력값을 확인 할 수 있습니다.

사용 신청하려 가기를 클릭하면 로그인 화면이 나타납니다. 로그인을 하고 사용 신청을 진행합니다.

Welcome to Kiwoom!

☞ 로그인하시면 키움증권이 제공하는 다양한 서비스와 함께 편리하게
사이트를 이용하실 수 있습니다.



<p>인증서 로그인</p> <p>❶ 공인인증서 로그인</p> <p><input type="checkbox"/> 공인인증서 자동 팝업 사용 <input checked="" type="checkbox"/> 주문서 자동 서명</p>	<p>ID 로그인</p> <p>• ID : <input type="text"/></p> <p>• ID 비밀번호 : <input type="password"/></p> <p>• 공인인증 비밀번호 : <input type="password"/></p> <p><input checked="" type="checkbox"/> 공인인증서 사용 <input checked="" type="checkbox"/> 주문서 자동서명 <input type="checkbox"/> 아이디 저장</p>
--	--

자동로그아웃 설정 자동로그아웃 사용안함

로그인을 하면 신청 화면이 나타납니다.

The screenshot shows the KiUM Open API+ service page. At the top, there's a navigation bar with links for 모바일상담, 원격지원, 소비자보호포털, 오픈스탁, 하우스탁, 클라우드펀딩, blog, and facebook. Below the navigation is a search bar labeled 'SEARCH'.

The main content area has a title '키움 Open API+' and a '즐겨찾기' button. A horizontal menu bar below the title includes '서비스 소개', '서비스 사용 등록/해지' (selected), 'FAQ', '자료실', and '고객문의 게시판'.

On the left, a sidebar lists services: 휴트레이딩시스템 (HTS), 스마트폰, 태블릿, Open API (selected), 키움 Open API+, 해외파생 Open API-W, API 마켓, ARS 서비스, and 기타자료실. A 'Quick Link' section at the bottom of the sidebar includes links for 영문문4, HTS 다운로드, 키움금융센터, and 모바일 가래조회.

The central content area contains several sections:

- 서비스 사용 등록/해지**: A note stating that using KiUM Open API services requires both the 'KiUM Open API Usage Terms' and the 'KiUM Open API User Agreement'.
- 키움 Open API 이용 유의사항**: A detailed note about transaction processing and user rights.
- 키움 Open API 이용 유의사항**: A large box containing the detailed note.
- 키움 Open API 사용자 계약서**: A note about the user agreement.
- 키움 Open API 사용자 계약서**: A large box containing the user agreement terms.
- 제1조(목적)**: A note about the purpose of the user agreement.
- 본인은 본 '키움 Open API 사용자 계약서' 내용을 충분히 이해하고 이에 동의 합니다.**: A checkbox for accepting the user agreement.
- 서비스 사용 등록**: A button at the bottom of the acceptance section.
- 처리결과**: A placeholder for the result of the registration process.
- ① '키움 Open API 사용 등록' 시 등록한 고객 보유 모든 ID로 '키움 Open API' 사용이 가능합니다.**: A note about using multiple IDs for login.
- 사용자 계약서 및 유의사항 변경 골자를 위한 이용자 동의사항**: A note about changes to the user agreement and terms of use.

서비스 사용 등록을 클릭해서 사용 신청을 진행합니다.

등록을 마쳤다면 이용에 동의하고 등록하기를 클릭합니다.

ARS서비스 기타자료실	<p>1. 고객에게 제공되는 TR(transaction)과 보물은 당사의 시스템 운영 시 지속적으로 변경될 수 있습니다. 변경 내용은 당시 '키움 Open API' 계약 판을 통하여 공지하며, 사용 고객님들께서는 항상 계약판을 확인하여 주시기 바랍니다. 내용 변경 시, 변경 내용을 프로그램에 직접 반영하셔야 하며, 미 반영으로 인한 프로그램 실행 오류 등은 달사가 책임을 지지 않습니다. 달 서비스 는 불특정 다수를 대상으로 제공하는 서비스로 시스템 변경 및 오류 등에 대하여 고객님들께 개별 연락을 드리지 못하는 점 양지하여 주시기 바랍니다.</p> <p><input type="checkbox"/> 본인은 본 '키움 Open API 이용 유의사항' 내용을 충분히 이해하고 이에 동의 합니다.</p>								
Quick Link 영문문4 HTS 다운로드 키움금융센터 모바일 거래조회	<p>키움 Open API 사용자 계약서 다운로드(PDF)</p> <p>키움 Open API 사용자 계약서</p> <p>본 계약은 키움증권(이하 "회사"라 한다)과 키움 Open API 사용자(이하 "사용자"라 한다)간의 키움 Open API 서비스(이하 "서비스"라 한다) 사용 에 관한 계약이다. 본 계약의 내용에 동의하여, 동의 버튼을 누르는 것은 사용자가 본 계약서의 내용에 이의가 없음을 의미한다.</p> <p>제1조(목적)</p> <p><input type="checkbox"/> 본인은 본 '키움 Open API 사용자 계약서' 내용을 충분히 이해하고 이에 동의 합니다.</p> <p>서비스 사용 등록</p> <p>처리결과 등록이 완료되었습니다.</p> <p>① '키움 Open API 사용 등록' 시 등록한 고객 보유 모든 ID로 '키움 Open API' 사용이 가능합니다.</p> <p>■ 사용자 계약서 및 유의사항 변경 공지를 위한 이용자 동의사항</p> <ul style="list-style-type: none"> 개인정보의 수집 이용 목적 - 사용자 계약서 및 유의사항 변경 공지 수집하는 개인정보의 항목 - 이메일주소, 휴대폰번호 개인정보의 보유 및 이용기간 - 제공받은 목적 달성을 후 즉시 파기 (서비스 해지 시 삭제) *서비스 해지 후 재가입시에는 다시 동의하세요만, 변경사항 공지를 받으실 수 있습니다. 개인정보 수집이용에 동의하지 않으실 수 있습니다. 동의하지 않으셔도 서비스 사용등록은 가능하며, 계약서 및 유의사항 변경사항은 고객문의계시판 공지사항을 통해서만 확인 가능합니다. <p>위와 같이 개인정보를 수집 이용하는데 동의 하십니까?</p> <p>동의함 <input checked="" type="radio"/> 동의하지 않음 <input type="radio"/></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">E-Mail</td> <td>jswoo030@gmail.com</td> <td style="width: 30%;">SMS</td> <td>010-5039-****</td> </tr> <tr> <td colspan="2" style="text-align: right; padding-right: 10px;">고객정보 변경</td> <td colspan="2" style="text-align: right; padding-right: 10px;">등록하기</td> </tr> </table>	E-Mail	jswoo030@gmail.com	SMS	010-5039-****	고객정보 변경		등록하기	
E-Mail	jswoo030@gmail.com	SMS	010-5039-****						
고객정보 변경		등록하기							

아래 그림은 등록이 완료된 화면입니다.

The screenshot shows the KIUM Open API+ service registration confirmation page. At the top, there is a navigation bar with links for mobile support, member support, consumer protection, open API, futures, FX, asset management, online inquiry, search, and broadcast/education. Below the navigation bar, there is a sidebar with links for Home Trading System (HTS), smartphone, tablet, Open API, API Market, ARS services, and other resources. The main content area is titled "KIUM Open API+" and contains tabs for Service Overview, Registration/Deactivation, FAQ, Resources, Customer Support, and Development Guide. The "Registration/Deactivation" tab is selected, showing a table with one row of data: "Service Registration Date" (2013/05/23), "ID" (redacted), "Customer Name" (우종선), and "Deactivation" (button). Below the table, there is a section titled "User Agreement and Privacy Policy Changes" with several bullet points. At the bottom, there is a question "Would you like to receive personal information collection notices?" with two radio button options: "Yes" and "No". A footer at the bottom includes fields for E-Mail (jewoo030@gmail.com), SMS (010-5039-****), and buttons for "Change Information" and "Register".

2.2 API 설치하기 환경설정

2.2절에서는 API를 설치하고 환경설정을 하는 방법에 대해 다릅니다.

API를 설치하기 위해서는 먼저 키움증권 홈페이지의 Open API페이지에 들어갑니다.

API 다운로드 홈페이지 링크

기타자료실 영문문4(HTS)와 동일

■ **키움 Open API+ 사용 요건**
당사 계좌를 보유하고 HTS ID를 연결하신 고객님께서는 모두 이용 가능하며, '서비스 사용등록' 탭에서 사용 등록 후 바로 이용 가능합니다.

■ **키움 Open API+ 사용절차**

Quick Link

영문문4
HTS 다운로드
기융금융센터
모바일 거래조회

Step 1 Open API 서비스를 사용 신청합니다 → **Step 2** Open API 설치 → **Step 3** OCX 탑재 프로그램 제작 → **Step 4** 테스트 및 디버깅

사용 신청하러 가기 > 키움 Open API+ 모듈 다운로드 <--> 개발가이드 바로가기 <--> 상시 모의투자 신청하러 가기 >

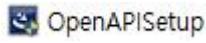
KOA Studio 다운로드 >

교육 VOD 바로가기 >

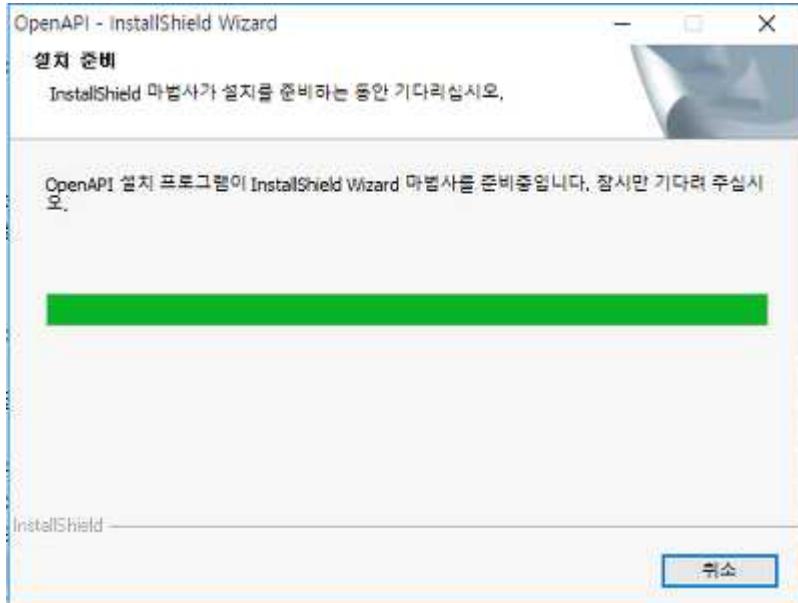
자료실 바로가기 >

■ STEP 1. Open API 사용신청 합니다.
- 키움 Open API+ 서비스 이용을 위하여는 사용 등록 신청이 필요합니다.
- 당사 '홈페이지 > 트레이딩 체널 > Open API > 키움 Open API+ > 서비스 사용 등록/해제' 탭에서 사용등록이 가능합니다.

"키움 Open API+ 모듈 다운로드"를 클릭하고 다운로드를 진행합니다. 다운로드를 완료하면 아래 그림과 같은 아이콘이 생성됩니다.



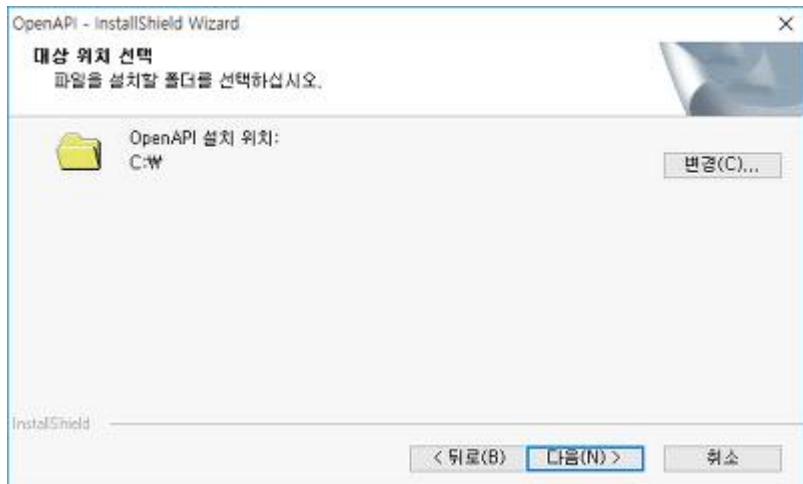
아이콘을 클릭하고 설치를 진행합니다.



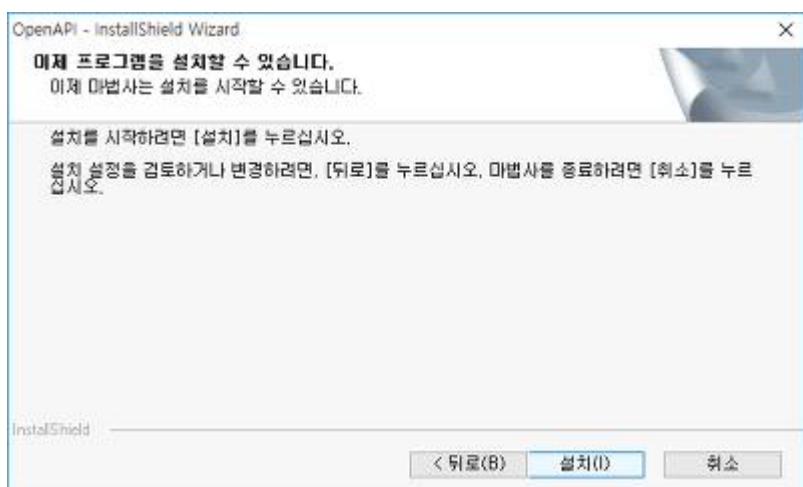
다음을 클릭합니다.



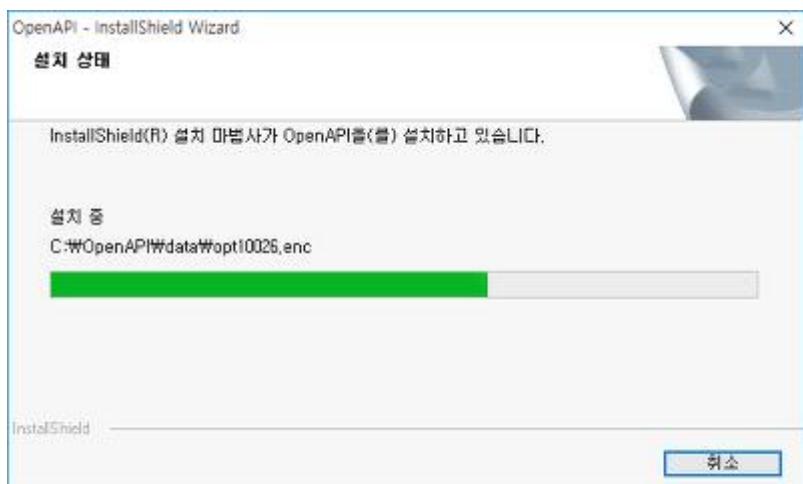
설치 경로를 지정합니다. 설치 경로를 선택 할 수 있지만, 앞으로 책의 내용을 원활히 따라하기 위해서 경로를 변경하지 않고 다음을 클릭합니다.



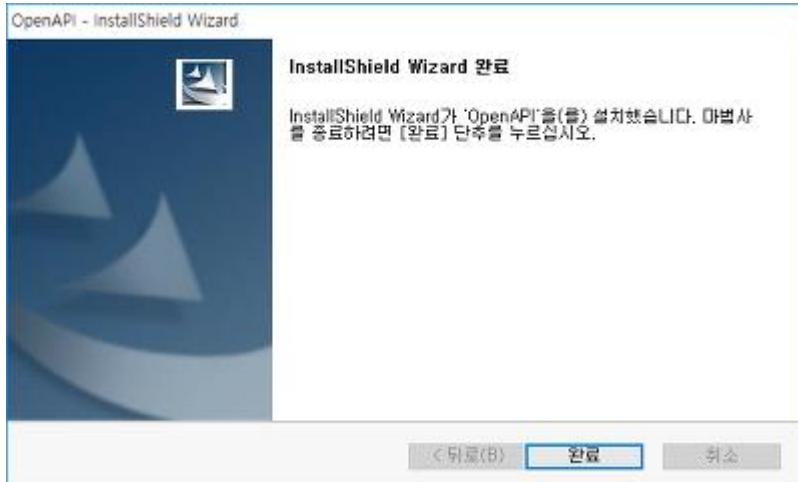
다음을 클릭합니다.



설치를 클릭합니다.



완료를 클릭합니다.



설치가 완료되면 C:\OpenAPI 에 khopenapi.ocx 가 있습니다.

내 PC > 로컬 디스크 (C:) > OpenAPI >				
이름	수정한 날짜	유형	크기	
kdfapi2.dll	2017-08-...	응용 프로그램 확장	4,159KB	
kdfnj.dll	2017-08-...	응용 프로그램 확장	3,970KB	
khcrypt.dll	2016-06-...	응용 프로그램 확장	24KB	
khcryptcore.dll	2016-11-...	응용 프로그램 확장	75KB	
khcryptex.dll	2016-11-...	응용 프로그램 확장	20KB	
khopenapi.ocx	2018-04-...	ActiveX 컨트롤	424KB	
klib.dll	2017-08-...	응용 프로그램 확장	557KB	
koa_devguide.xml	2018-02-...	XML 문서	62KB	
koascreentrmap.ini	2017-08-...	구성 설정	22KB	

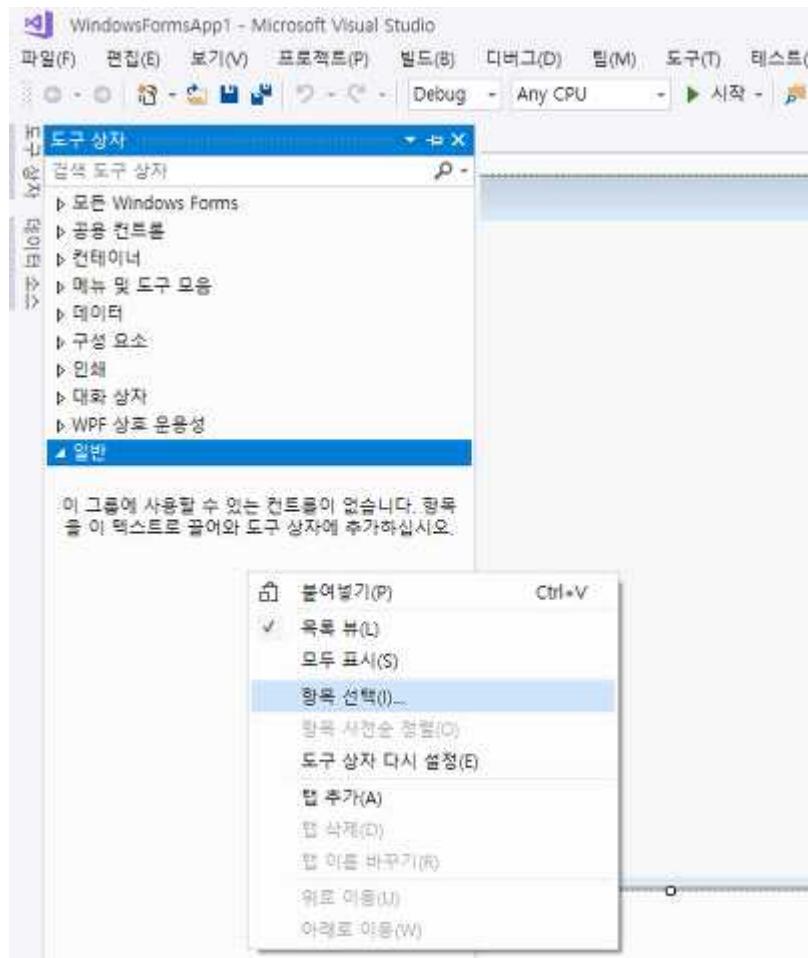
이 Active X 컨트롤을 화면에 추가해서 자동매매 프로그램을 만들게 됩니다.

이제 Visual Studio를 실행하고 설치한 Open API를 사용 할 수 있도록 환경설정을 해보겠습니다.

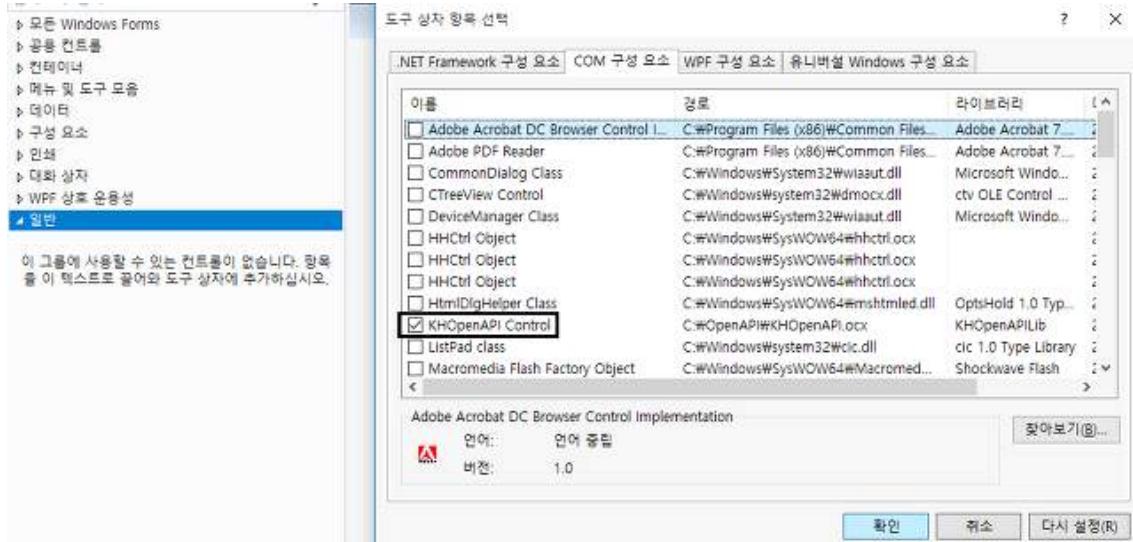
Open API를 사용하기 위해서는 도구 상자에 khopenapi.ocx 컨트롤을 추가합니다. 그리고 화면에 버튼처럼 가져다 놓습니다.

먼저 Windows Forms 앱 프로젝트를 생성하고 도구상자를 클릭합니다.

[일반]을 클릭하고 마우스 오른쪽 클릭 후 항목 선택을 클릭합니다.



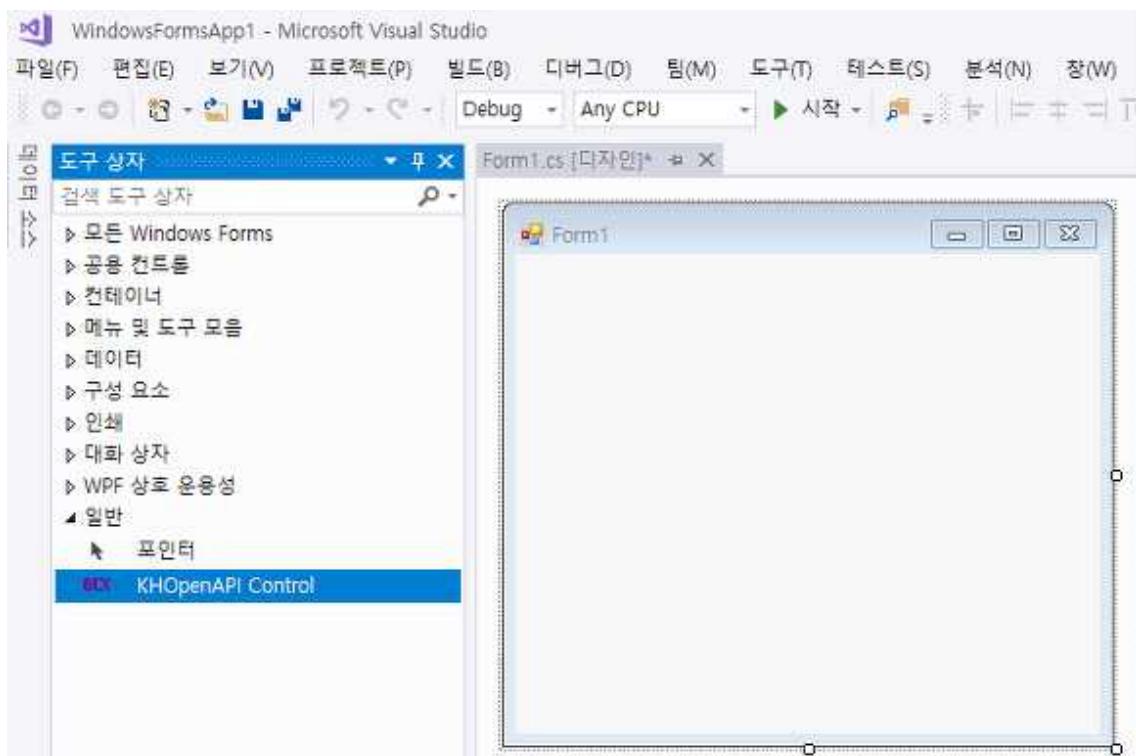
"도구 상자 항목 선택" 창이 나타납니다.

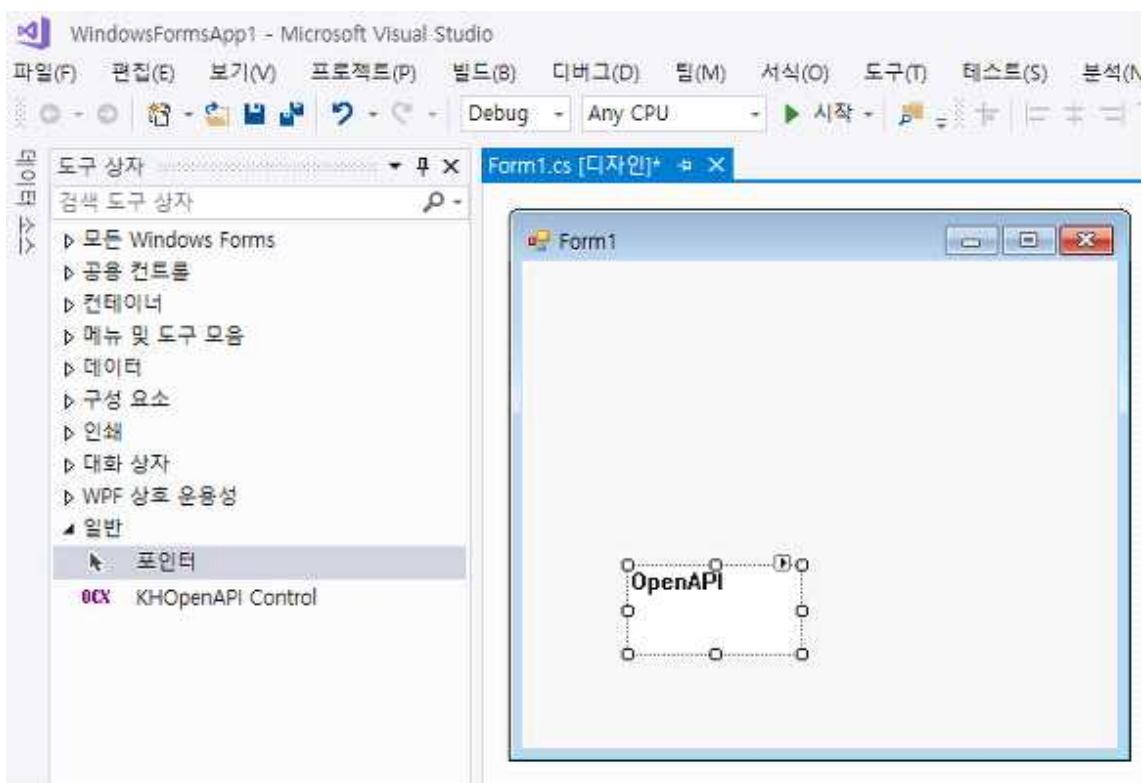


COM 구성요소 탭을 클릭하고 KHOpenAPI Control 을 체크합니다.

체크를 완료했으면 확인을 클릭합니다.

이제 KHOpenAPI Control을 드래그해서 Form 화면으로 추가합니다. 버튼을 드래그 해서 가져다 놓는 것과 같습니다.

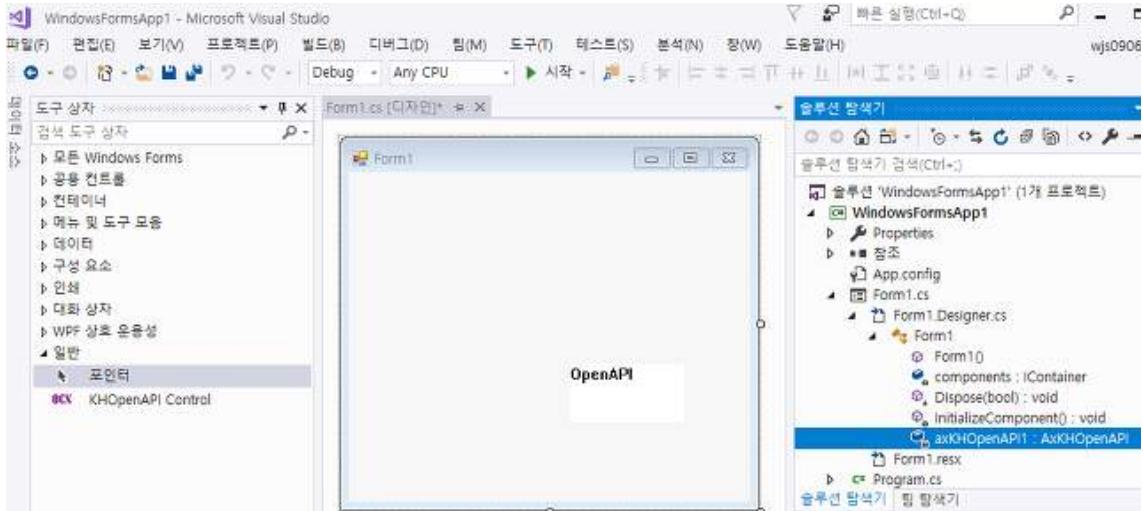




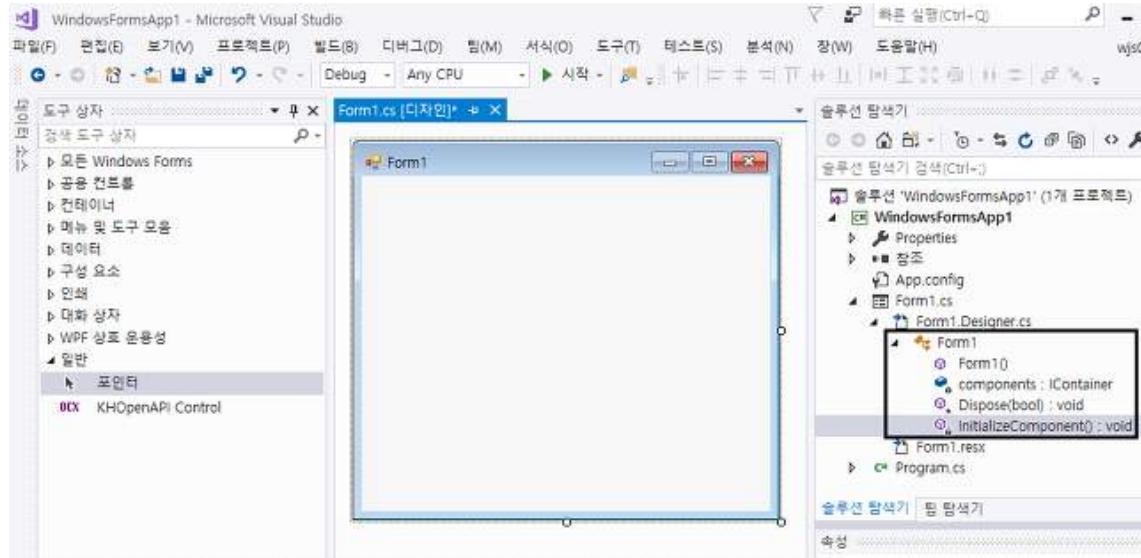
화면에 추가 된 Control을 사용해서 키움증권 API에서 제공하는 함수들을 호출 할 수 있습니다.

프로젝트의 솔루션 탐색기를 보시면 Control을 추가하기 전과 후의 차이점을 알 수 있습니다.

아래 그림은 Control을 Form1의 화면에 추가하고 난 후에 솔루션 탐색기를 확인한 화면입니다.



Control을 추가하고 나면 axKHOpenAPI1이 솔루션 탐색기에 나타납니다.

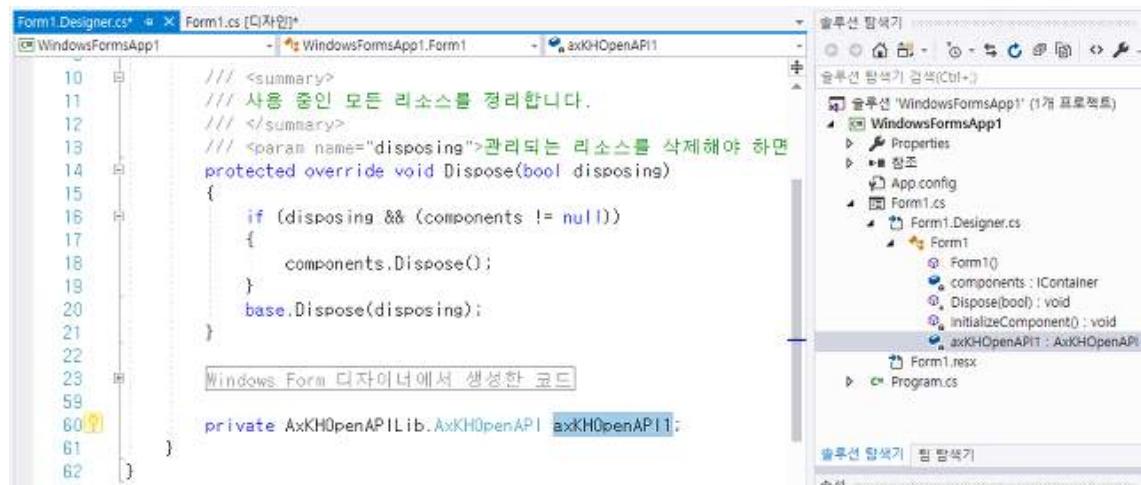


Control을 추가하기 전에는 axKHOpenAPI1이 목록에 없는 모습입니다.

axKHOpenAPI1은 앞으로 키움증권 API의 함수를 호출하기 위해서 쓰일 참조변수의 이름입니다.

그림처럼 솔루션 탐색기에 추가된 내용은 Form1.cs에서 접근하고 사용 할 수 있습니다.

axKHOpenAPI1을 더블클릭하면 Form1.cs에 변수가 추가된 것을 확인 할 수 있습니다.



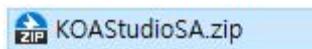
앞으로 3.1장부터 이 변수를 사용해서 Form1()에 소스코드를 작성해보도록 하겠습니다.

2.3 KOA Studio 설치하기

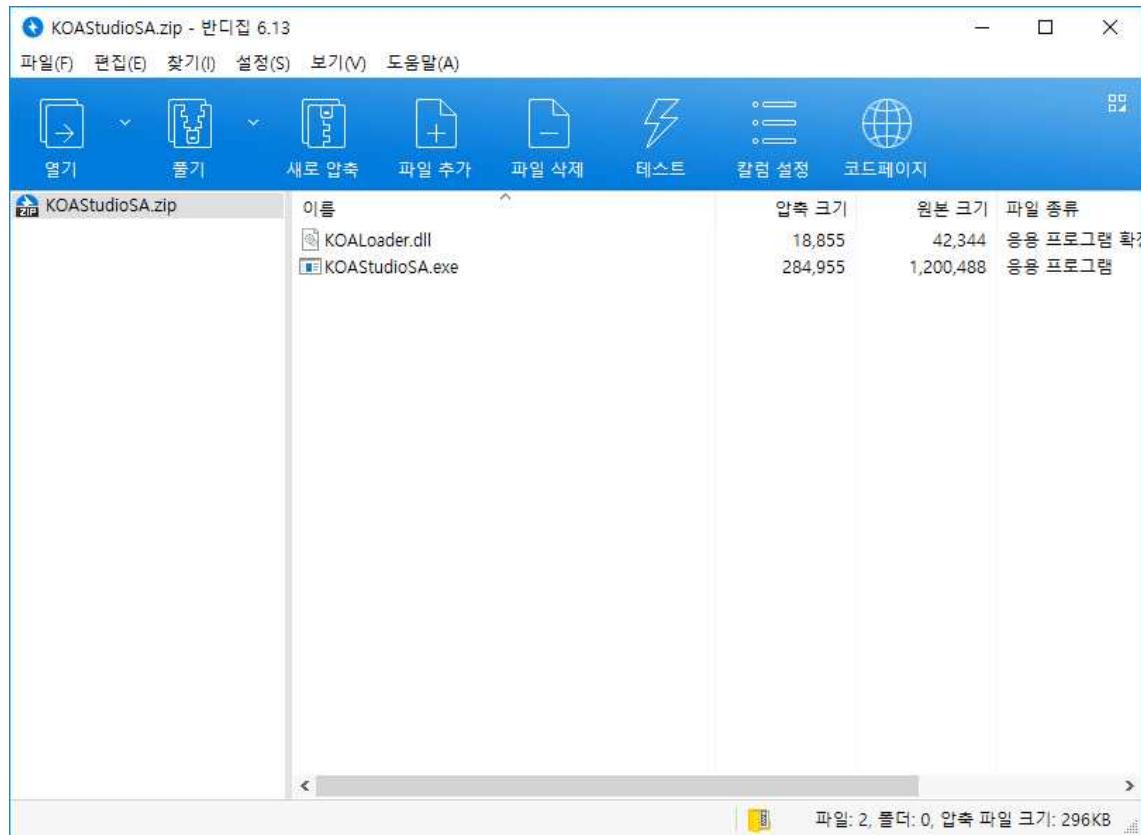
앞서 1.2장에서 잠시 소개하였듯이 KOA Studio는 키움 API의 사용 설명서입니다. 앞으로 자동매매 프로그램을 만들 때 참고하기 위해서 아래 주소에서 다운로드 하도록 합니다.

<https://www1.kiwoom.com/nkw.templateFrameSet.do?m=m1408000000>

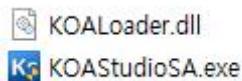
KOA Studio 다운로드를 클릭하면 .zip 파일을 다운로드 받을 수 있습니다.



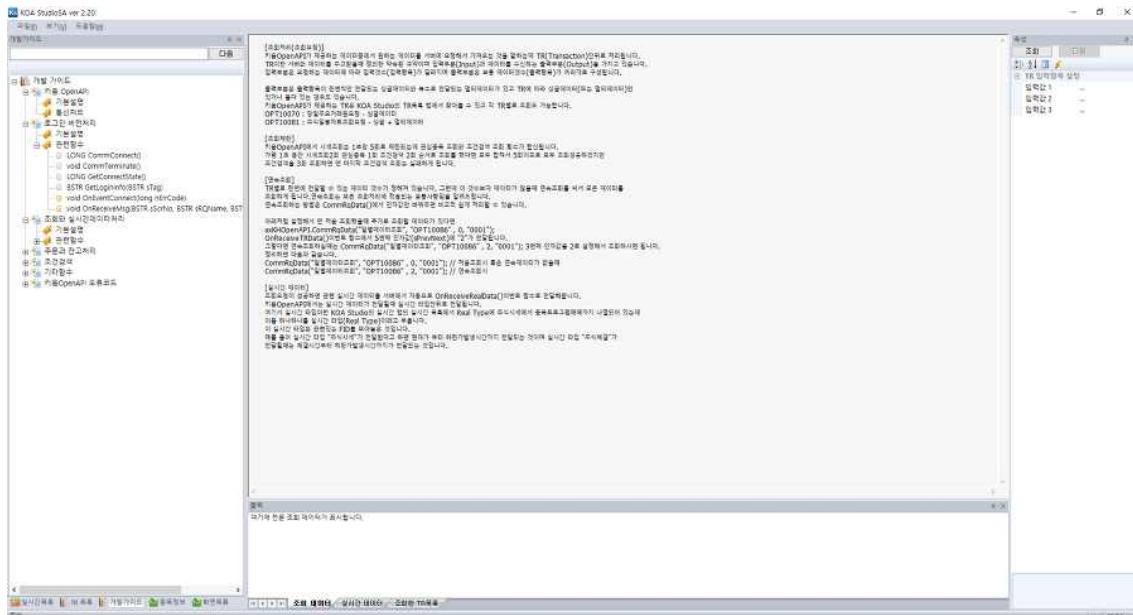
파일의 압축을 해제합니다.



압축을 해제하면 KOASTudioSA.exe 파일이 나타납니다.



앞으로 KOAStudioSA.exe 파일을 참고해서 자동매매 프로그램을 만들게 될 것입니다.



3. 사용자 로그인

Preview

3장에서는 로그인 사용자의 정보를 확인 하는 방법을 소개합니다. 프로그램에서 사용자의 정보를 확인 하는 것은 중요합니다. 앞으로 프로그램에서 종목을 주문 할 때 사용자의 정보가 필요하기 때문입니다. 3장을 잘 학습해서 프로그램에서 사용자의 정보를 쉽게 확인하도록 합니다.

Contents

3.1장에서는 프로그램에서 hts처럼 로그인합니다. 로그인 화면을 띄우는 CommConnect() 함수에 대해서 알아봅니다.

3.2장에서는 사용자 정보를 확인합니다. 사용자 로그인시 호출되는 OnEventConnect 이벤트 함수에 대해서 알아봅니다.

3.3장에서는 로그인 사용자의 계좌 잔고를 알아봅니다. 서버에 정보를 요청하는 CommRqData 함수에 대해서 알아봅니다. OnReceiveTrData 이벤트 함수에 대해서 학습합니다. TR data가 무엇인지 알아봅니다.

3.1 프로그램에서 HTS처럼 로그인하기

hts에서 주식을 주문하려면 로그인을 해야 합니다. 사용자의 프로그램에서도 HTS처럼 로그인을 해야 주식을 주문 할 수 있습니다. 3.1장에서는 사용자의 프로그램에서 HTS처럼 로그인 하는 방법에 대해서 알아봅니다. 3.1장의 내용을 학습한다면 아래 그림처럼 로그인 화면을 띠울 수 있습니다.



3.1 자동매매 프로그램에서 영웅문 HTS처럼 로그인하기

먼저 1.3장에서 만든 것처럼 Window Forms 앱(.NET Framework) 프로젝트를 생성합니다. 2.2장에서 OCX Control을 잘 추가했다면, 키움 증권 API 안의 함수들을 사용하기 위한 참조변수를 사용 할 수 있을 것입니다. 1.3장에서 프로젝트의 Form1() 파일에 코드를 작성한다고 설명했었습니다.

Window Forms 앱(.NET Framework) 프로젝트를 생성하면 Form1()의 코드가 아래 코드처럼 나타납니다.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

수많은 using으로 시작하는 문장들은 검은색 또는 회색을 띕니다. using 뒤의 항목들은 현재 프로젝트에서 기능을 참조해서 사용하는 항목들입니다. 현재 프로젝트에서 사용하고 있는 기능은 글씨가 검은색으로 바뀝니다. 사용하지 않는 기능은 글씨가 회색으로 표시됩니다.

여기서 앞으로 주목해야 할 부분은 namespace WindowsFormsApp1 { } 의 중괄호 안에 있는 코드를 입니다. 앞으로 namespace WindowsFormsApp1 { } 의 중괄호 안에 코드를 작성하게 됩니다. namespace 옆의 WindowsFormsApp1은 현재 생성한 프로젝트의 이름입니다. public partial class Form1 : Form 은 현재 클래스가 화면 Form에 관한 것임을 의미합니다.

아래 코드는 생성자라고 부릅니다. 프로그램을 시작 하게 되면 화면이 나타나고 프로그램이 실행 됩니다. 생성자는 화면을 생성하는 역할을 합니다.

생성자

```
public Form1()
{
    InitializeComponent();
}
```

이제부터 Form1.cs에 어떻게 코드를 작성해야 로그인 화면을 띄울 수 있는지 살펴보도록 하겠습니다. 3.1장에서 만들게 될 예제 프로그램은 실행되면서 로그인 화면을 띄워야 합니다. Form1()의 생성자에 작성된 코드들은 화면이 생성되면서 실행됩니다. 그렇다면 생성자에 로그인 화면을 띄우는 코드를 작성하면 됩니다.

Open API 에서는 로그인 화면을 띄우는 함수를 제공해주고 있습니다. 로그인 화면을 띄우는 함수의 이름은 CommConnect입니다. OCX 컨트롤을 추가하면서 생성된 변수 axKHOpenAPI1를 사용해서 함수를 호출 할 수 있습니다. 자세한 함수의 호출 방법을 알기 위해서 아래 코드를 보겠습니다.

```
axKHOpenAPI1.CommConnect();
```

axKHOpenAPI1 처럼 함수를 호출 할 수 있게 해주는 변수를 "참조변수" 라고 부릅니다. 참조변수의 뒤에 . 을 찍고 OCX 컨트롤의 함수들을 호출 할 수 있습니다.

결국 아래와 같이 Form1.cs의 생성자에 코드를 작성하면 로그인 화면을 띄울 수 있습니다.

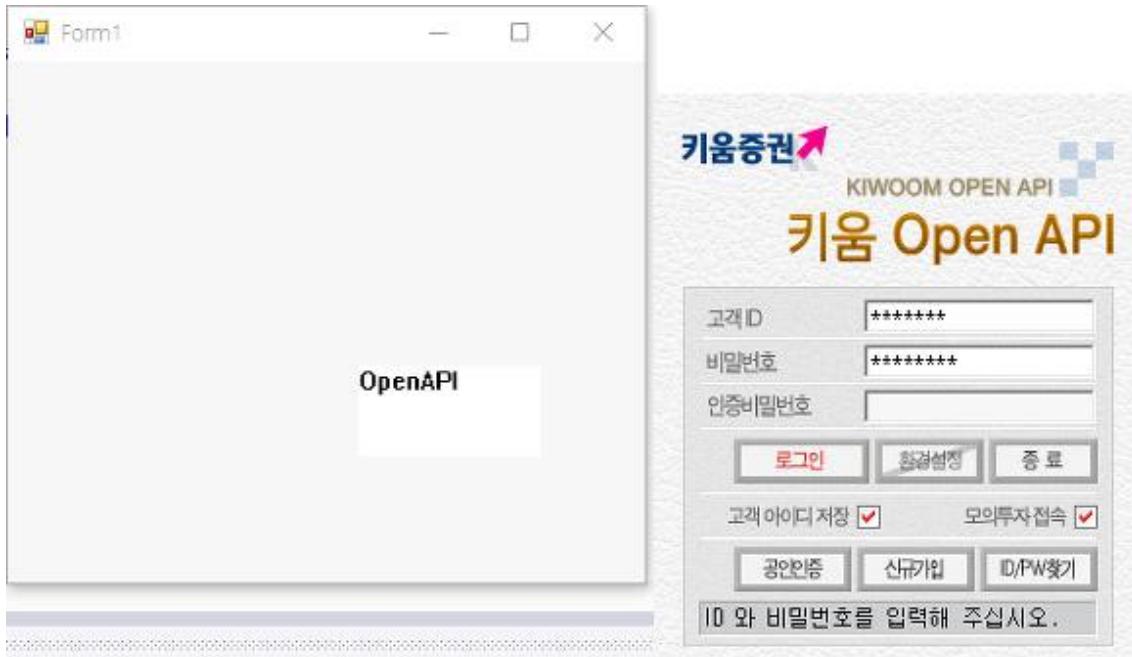
전체 코드

```
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            axKHOpenAPI1.CommConnect();
        }
    }
}
```

앞으로도 HTS 화면 속의 기능을 구현하기 위해서는 OCX Control의 참조 변수를 사용해서 API 함수를 호출하면 됩니다.

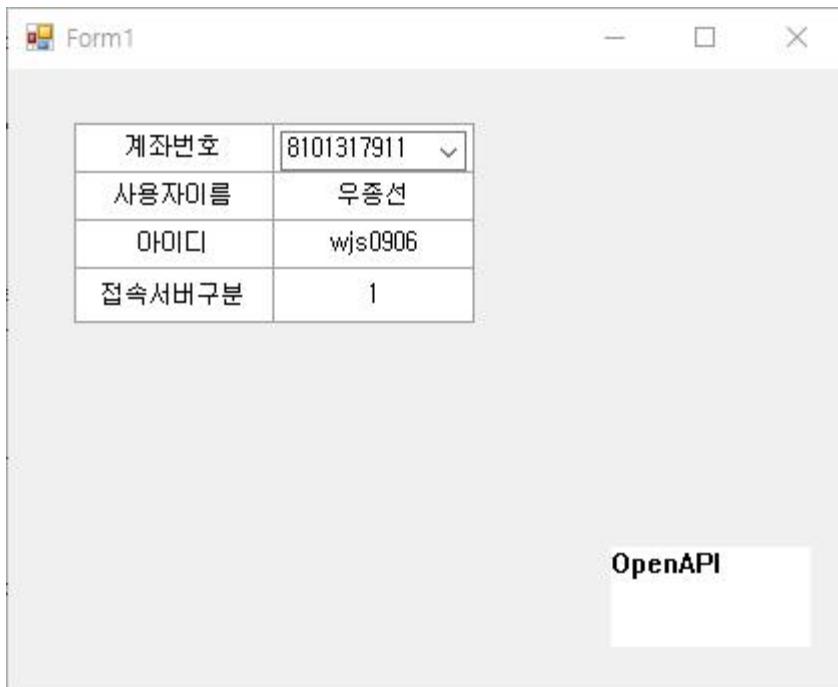
여기까지 입력하고 코드를 실행하면 아래 그림처럼 로그인 화면이 나타나는 것을 확인 할 수 있습니다. 로그인 윈도우에서 HTS처럼 비밀번호를 입력하고, 로그인 버튼을 클릭하면 로그인이 진행됩니다.



이 코드는 앞으로 만들 자동매매 프로그램 예제에서 항상 먼저 작성해야 하는 코드입니다. 함수를 통해서 로그인 윈도우를 띄우는 데 성공했을 때는 함수의 리턴 값으로 사용자에게 0을 반환하고, 로그인 윈도우를 띄우는 데 실패했을 때는 음수 값을 반환합니다. 로그인이 성공하거나 실패하면 OnEventConnect 이벤트가 발생하고 OnEventConnect 이벤트 함수의 인자 값으로 로그인 성공 여부를 알 수 있습니다. 이벤트 함수가 무엇인지에 대해서는 3.2장에서 더 알아보도록 하겠습니다.

3.2 로그인 사용자 정보 확인

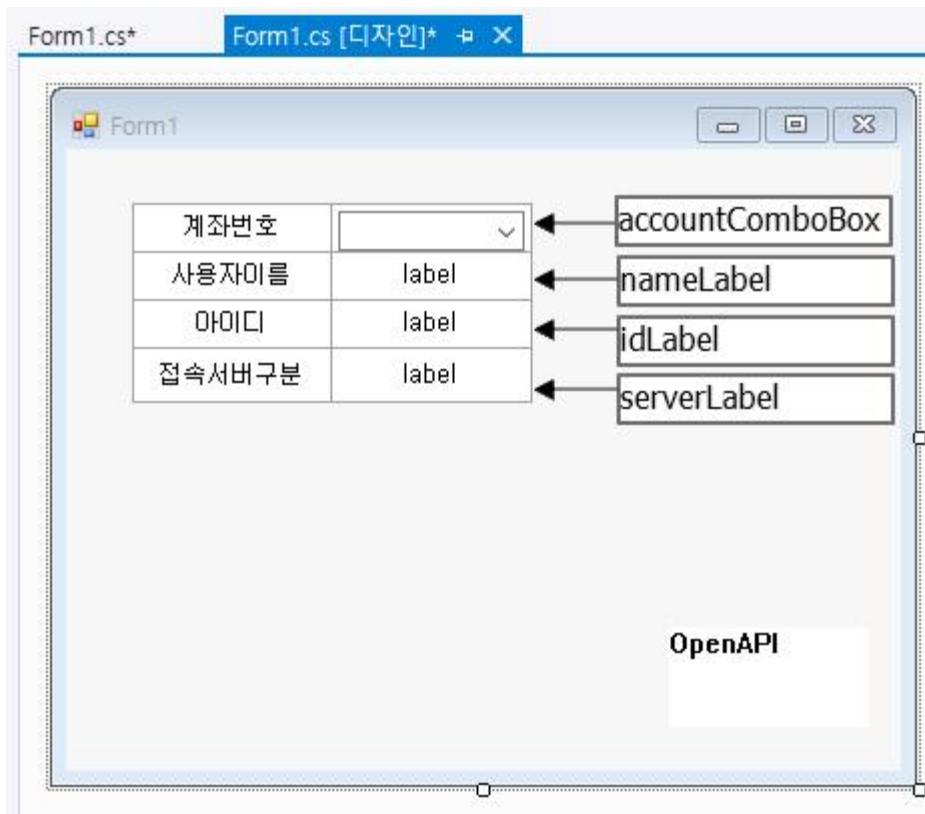
3.1장에서는 로그e인 윈도우를 출력하고 HTS 화면에서처럼 로그인 하는 방법에 대해서 알아보았습니다. 3.2장에서는 프로그램에서 로그인 한 사용자의 정보를 프로그램 화면에 출력해서 사용자 정보를 확인하는 방법에 대해서 알아보도록 하겠습니다. 3.2장의 내용을 학습하면 아래 그림처럼 사용자 정보를 프로그램 화면에 출력 할 수 있습니다.



먼저 프로그램의 화면을 만듭니다.

화면을 구성하는 순서는 다음과 같습니다.

- TableLayout을 배치하고 행을 추가합니다.
- label들을 배치하고 label 속성을 변경합니다.
- ComboBox , TextBox를 가져다 놓습니다.



Form1.cs 화면의 TableLayoutPanel 속성입니다.

속성	값
BackColor	White
CellBorderStyle	Single

Form1.cs 화면의 accountComboBox 속성입니다.

속성	값
Anchor	Top, Bottom, Left, Right
(Name)	accountComboBox

Form1.cs 화면의 nameLabel 속성입니다.

속성	값
Anchor	Top, Bottom, Left, Right
(Name)	nameLabel

Text	label
TextAlign	MiddleCenter

Form1.cs 화면의 idLabel 속성입니다.

속성	값
Anchor	Top, Bottom, Left, Right

(Name)	idLabel
Text	label

속성	값
Anchor	Top, Bottom, Left, Right

(Name)	serverLabel
Text	label

속성	값
Anchor	Top, Bottom, Left, Right

(Name)	serverLabel
Text	label

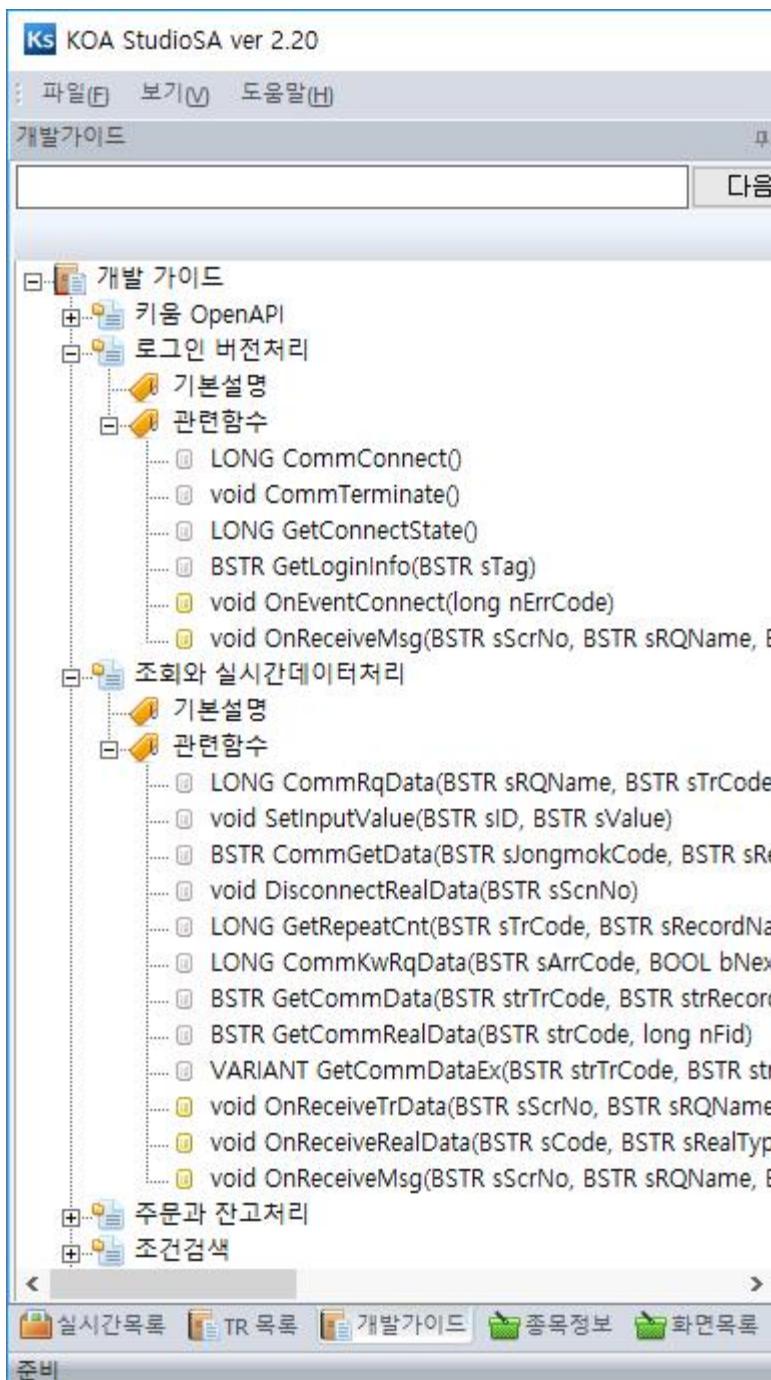
속성	값
Anchor	Top, Bottom, Left, Right

(Name)	serverLabel
Text	label

속성	값
Anchor	Top, Bottom, Left, Right

화면을 모두 구성 한 후 코드를 작성합니다.

앞서 3.1장에서 CommConnect 함수를 호출하면 OnEventConnect 이벤트가 발생한다고 배웠습니다. 이벤트 함수는 CommConnect 함수 같은 일반 함수가 호출 되었을때 호출 되는 함수입니다. KOA Studio를 참고하면 키움 API에서 제공하는 함수 중에서 어떤 함수들이 일반 함수이고, 어떤 함수들이 이벤트 함수인지 알 수 있습니다.



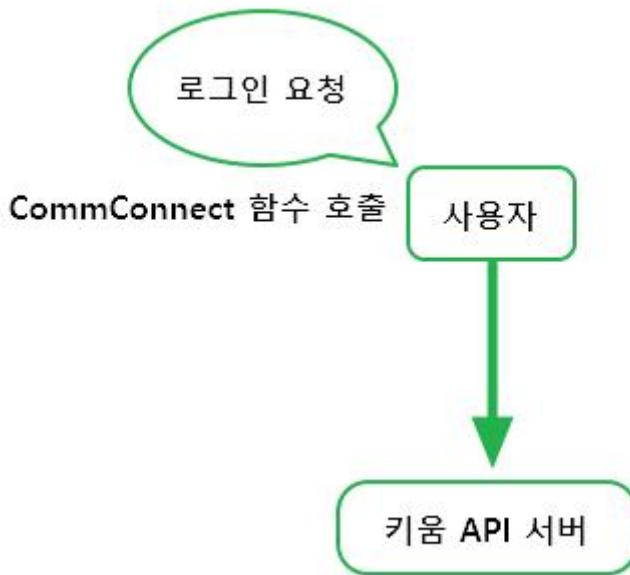
KOA Studio를 보면 키움 API에서 어떤 함수들을 제공하고 있는지 알 수 있습니다. 함수의 이름 앞에 흰색 아이콘이 있는 것은 일반 함수입니다. 참조변수 axKHOpenAPI1를 사용해서 호출하고 사용하면 됩니다. 함수의 이름 앞에 노란색 아이콘이 있는 것은 이벤트 함수입니다.

사용자가 일반 함수를 호출하면 이벤트 함수가 호출된다고 이야기 했었습니다. 하지만 CommConnect 함수를 호출한다고 해서 OnReceiveTrData 함수가 호출되지는 않습니다. 같은 풀더 끝에 있는 이벤트 함수가 호출됩니다. 예를 들어서 CommRqData 함수를 호출하면 OnReceiveTrData 함수가 호출됩니다.

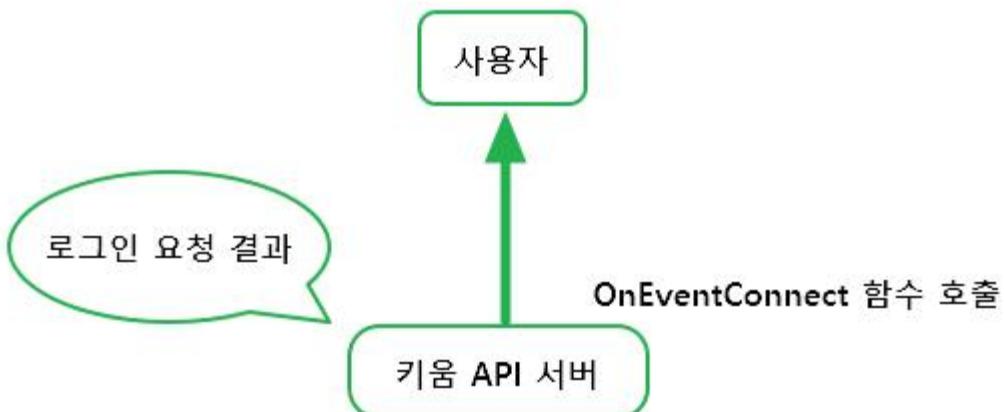
이렇게 일반함수가 이벤트 함수를 호출하는 방식은 사용자가 서버에 정보를 요청하는 방법을 말해줍니다. 아래 그림들을 통해서 정보를 요청하는 방법을 자세히 살펴보도록 하겠습니다.

다.

로그인 CommConnect 함수가 OnEventConnect 함수를 호출하는 과정을 살펴보겠습니다.



사용자는 CommConnect 일반함수를 호출해서 로그인 화면을 띄웁니다. 그리고 로그인을 진행합니다.



키움 API 서버는 사용자가 보낸 요청의 결과로 이벤트 함수를 호출합니다. 사용자는 이벤트 함수에서 일반함수로 보냈던 요청의 결과를 확인 할 수 있습니다. 요청의 결과는 이벤트 함수의 입력변수로 전달 됩니다.

코드를 함께 작성하면서 더 알아보도록 하겠습니다.

3.2장의 코드 작성 순서는 아래 목록과 같습니다.

- 로그인 요청
- 호출 될 OnEventConnect 함수 작성

먼저 3.1장에서 했던 것처럼 로그인 화면을 띄웁니다.

CommConnect 함수로 로그인 요청

로그인 요청이 발생하면 프로그램에서 로그인 원도우를 출력합니다. 그리고 프로그램에서 로그인이 성공하거나 실패하면 OnEventConnect 이벤트 함수가 호출됩니다.

OnEventConnect 이벤트 함수 호출

OnEventConnect 이벤트 함수는 프로그램의 소스코드에서 아래와 같이 정의됩니다.

```
public void OnEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e){
    .....
    .....
}
```

OnEventConnect 이벤트 함수가 호출되면 사용자는 이벤트 함수의 입력변수 e를 통해서 로그인 성공여부를 알 수 있습니다. e.nErrCode의 값이 0이면 정상적으로 로그인 된 것입니다. 0이 아니면 정상적으로 로그인 되지 않은 것입니다.

그렇다면 OnEventConnect 함수 안에서 프로그램의 기능을 로그인이 성공 했을 때와 실패 했을 때로 나눠서 정의 할 수 있습니다. 로그인이 성공 했을 때, 사용자의 정보를 출력하도록 프로그램을 작성하면 됩니다. 사용자가 원하는대로 동작하도록 프로그램에서 호출될 OnEventConnect 함수의 기능을 새롭게 정의합니다.

먼저 Form1.cs의 생성자에 OnEventConnect() 함수를 추가합니다. 아래 코드를 Form1.cs의 생성자에 작성하면 됩니다.

```
axKHOpenAPI1.OnEventConnect += onEventConnect;
```

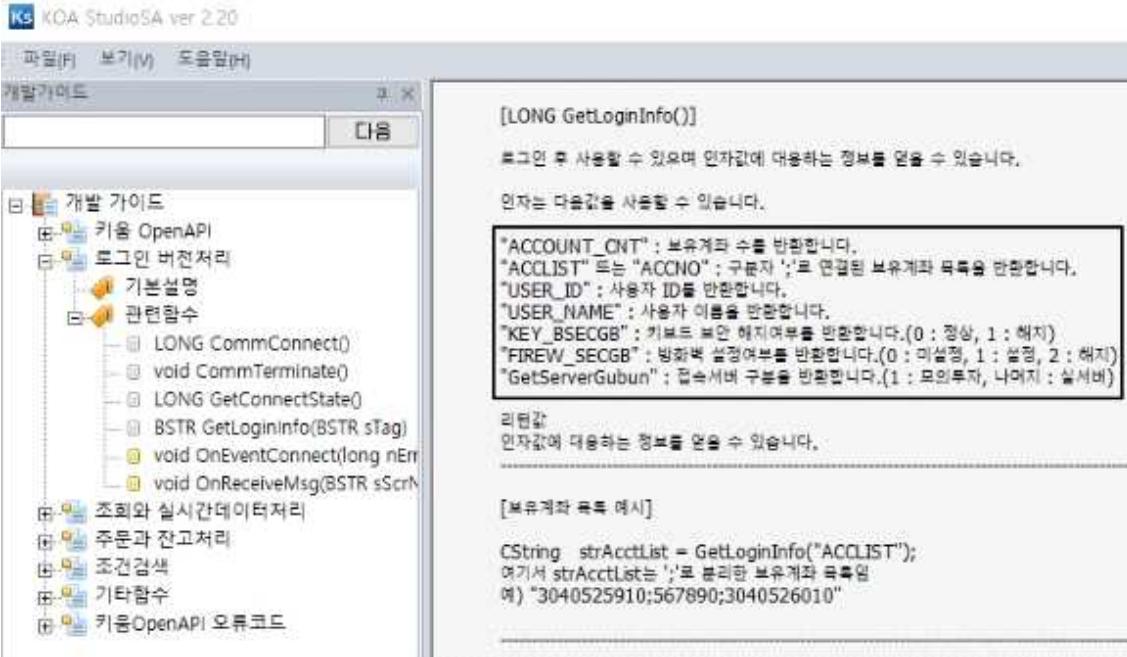
이렇게 코드를 작성하면 API의 OnEventConnect 함수에 사용자가 정의한 onEventConnect 함수의 내용을 덮어쓰게 됩니다.

onEventConnect 함수를 추가했다면 함수의 내용을 작성합니다

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    if (e.nErrCode == 0)
    {
        .
    }
}
```

onEventConnect 함수의 매개변수 e를 통해서 로그인 처리 결과를 확인 할 수 있습니다. 변수 e에 포함된 nErrCode 값이 0이면 정상적으로 로그인이 된 것입니다.

정상적으로 로그인이 되었을 때 API 함수를 호출해서 정보를 요청해보도록 하겠습니다. 요청의 결과로 반환된 정보는 화면의 ComboBox, Label들에 바인딩 하면 됩니다. KOA Studio를 참고해서 API 함수들을 호출해보도록 하겠습니다.



KOA Studio를 참고하시면 GetLoginInfo 함수를 사용해서 로그인 사용자 정보를 얻을 수 있습니다. 다음 코드는 사용자가 가지고 있는 계좌번호 목록을 요청하는 함수입니다.

```
axKHOpenAPI1.GetLoginInfo("ACCLIST");
```

계좌번호는 ; (세미콜론)이 포함된 문자열로 반환이 됩니다.

계좌번호 반환 예시 12345678;45678912;74185296

GetLoginInfo 함수의 괄호 안에 작성하는 문자열에 따라서 여러가지 다른 사용자 정보들을 요청 할 수 있습니다.

```
public void onEventConnect(object sender, AxKHOpenAPIlib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    if (e.nErrCode == 0)
    {
        string accountlist = axKHOpenAPI1.GetLoginInfo("ACCLIST");
        string[] account = accountlist.Split(';');
    }
}
```

string 타입의 accountlist에는 :(세미콜론)이 포함된 계좌번호 문자열이 담겨있습니다.

계좌번호만 추출하기 위해서는 ;을 기준으로 문자열을 분리해야합니다. C# 에서는 Split 함수를 사용해서 문자열을 분리 할 수 있습니다.

Split() 함수의 괄호 안에 ':'을 작성하면 ':'을 기준으로 분리합니다.

```
string[] account = accountlist.Split(':'');
```

분리된 문자열은 account 배열에 저장됩니다.

account 배열에 저장하고 for문을 사용해서 화면에 값을 바인딩 합니다.

```
public void onEventConnect(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    if (e.nErrCode == 0)
    {
        string accountlist = axKHOpenAPI1.GetLoginInfo("ACCLIST");
        string[] account = accountlist.Split(':' );
        for (int i = 0; i < account.Length; i++)
        {
            accountComboBox.Items.Add(account[i]);
        }
    }
}
```

account 배열에 있는 항목들을 한개씩 accountComboBox에 추가합니다. for문을 사용해서 comboBox에 항목을 추가하고 있습니다. .Length는 배열의 길이를 의미합니다.

사용자 이름 , 사용자 아이디 , 접속서버구분도 화면에 표시해보도록 하겠습니다.

사용자 이름은 GetLoginInfo() 함수에 "USER_NAME"를 전달하면 됩니다.

```
axKHOpenAPI1.GetLoginInfo("USER_NAME");
```

사용자 아이디는 GetLoginInfo() 함수에 "USER_ID"를 전달하면 됩니다.

```
axKHOpenAPI1.GetLoginInfo("USER_ID");
```

접속서버구분은 GetLoginInfo() 함수에 "GetServerGubun"을 전달하면 됩니다.

```
axKHOpenAPI1.GetLoginInfo("GetServerGubun");
```

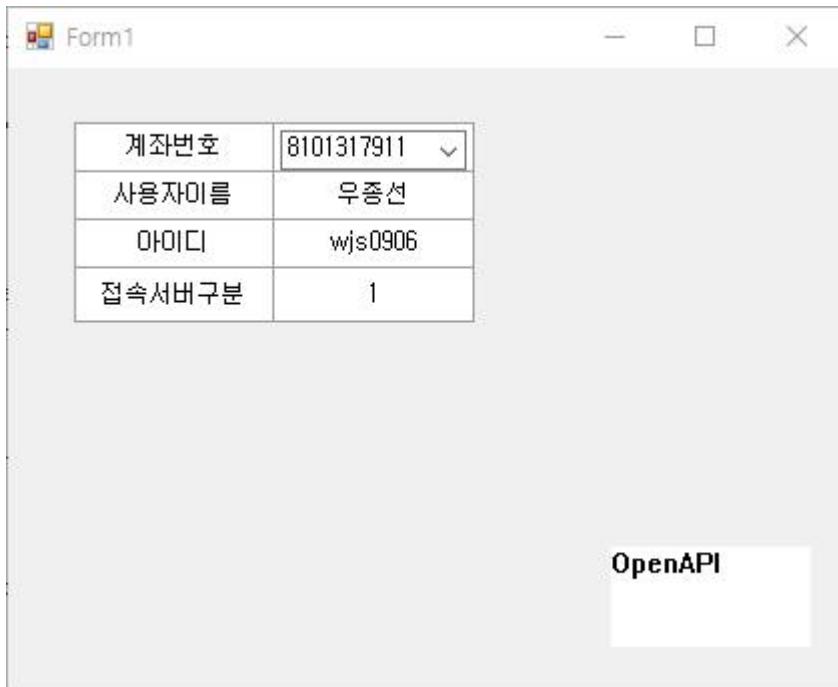
아래 코드는 Form1.cs의 전체 코드입니다. API 서버에 값을 요청하고 화면 label text에 바인딩했습니다.

```
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            axKHOpenAPI1.CommConnect();
            axKHOpenAPI1.OnEventConnect += onEventConnect;
        }

        public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
        {
            if (e.nErrCode == 0)
            {
                string accountlist = axKHOpenAPI1.GetLoginInfo("ACCLIST");
                string[] account = accountlist.Split(';');
                for (int i = 0; i < account.Length; i++)
                {
                    accountComboBox.Items.Add(account[i]);
                }
                string userId = axKHOpenAPI1.GetLoginInfo("USER_ID");
                string userName = axKHOpenAPI1.GetLoginInfo("USER_NAME");
                string connectedServer = axKHOpenAPI1.GetLoginInfo("GetServerGubun");
                idLabel.Text = userId;
                nameLabel.Text = userName;
                serverLabel.Text = connectedServer;
            }
        }
    }
}
```

실행화면

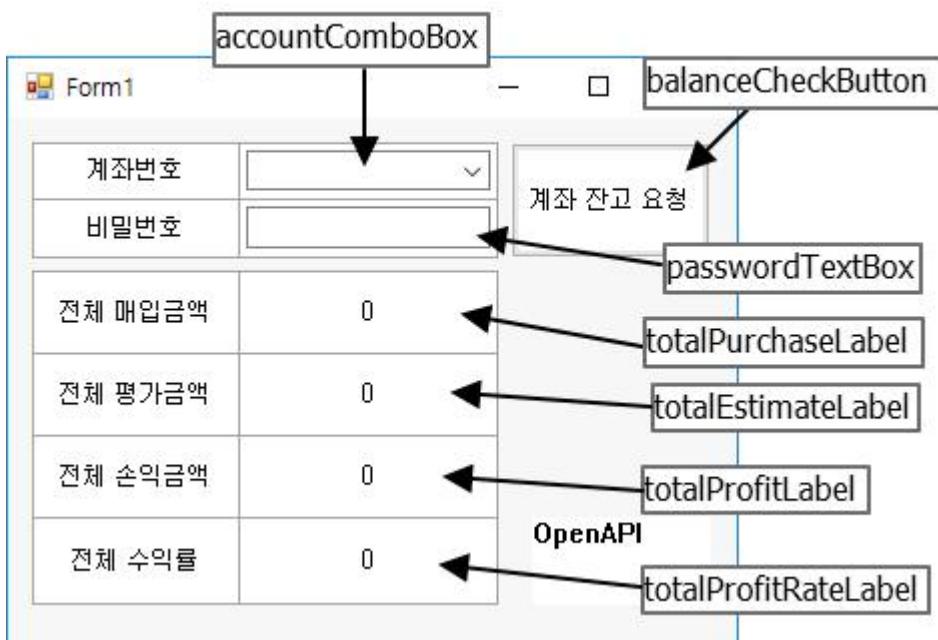
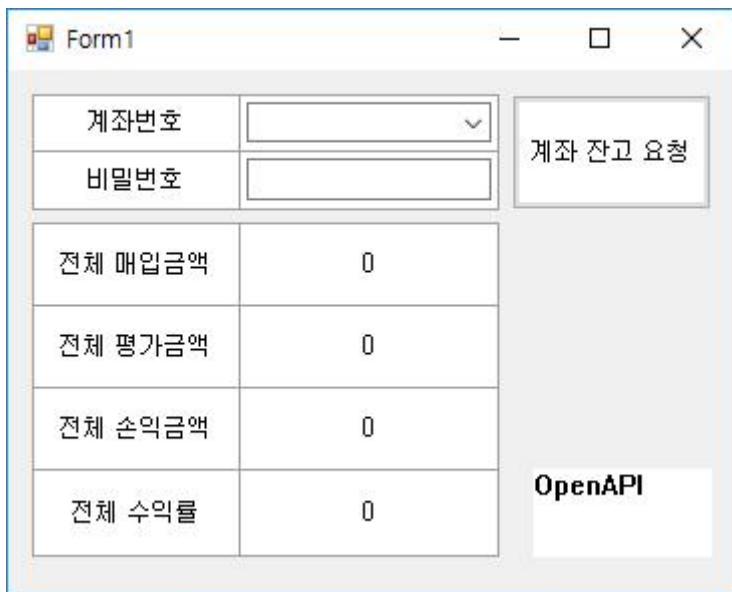


3.3장에서는 계좌잔고를 조회하는 방법에 대해서 알아보도록 하겠습니다.

3.3 사용자 계좌 조회

3.3장에서는 사용자 계좌 정보를 확인하는 방법에 대해서 알아보도록 하겠습니다. 3.3장에서 확인하게 될 정보는 사용자의 총매입금액, 평가금액, 손익금액, 수익률입니다.

먼저 프로그램의 화면을 구성합니다.



accountComboBox의 속성

속성 (Name)	값
Anchor	accountComboBox Top, Bottom, Left, Right

passwordTextBox의 속성

속성	값
(Name)	passwordTextBox
Anchor	Top, Bottom, Left, Right

totalPurchaseLabel의 속성

속성	값
(Name)	totalPurchaseLabel
Anchor	Top, Bottom, Left, Right
Text	0
TextAlign	MiddleCenter

totalEstimateLabel의 속성

속성	값
(Name)	totalEstimateLabel
Anchor	Top, Bottom, Left, Right
Text	0
TextAlign	MiddleCenter

totalProfitLabel의 속성

속성	값
(Name)	totalProfitLabel
Anchor	Top, Bottom, Left, Right
Text	0
TextAlign	MiddleCenter

totalProfitRateLabel의 속성

속성	값
(Name)	totalProfitRateLabel
Anchor	Top, Bottom, Left, Right
Text	0
TextAlign	MiddleCenter

balanceCheckButton의 속성

속성	값
(Name)	balanceCheckButton
Text	계좌 잔고 요청
TextAlign	MiddleCenter

화면 구성을 모두 마쳤다면 코드를 작성합니다. 코드의 작성 순서는 아래와 같습니다.

- 로그인
- 화면에 계좌번호 정보 바인딩
- 비밀번호 암호화
- 계좌 잔고 요청

로그인부터 코드를 작성해보도록 하겠습니다. Form1.cs의 생성자에 코드를 추가합니다.

```
public Form1()
{
    InitializeComponent();
    axKHOpenAPI1.CommConnect();
}
```

- 로그인
- 화면에 계좌번호 정보 바인딩
- 비밀번호 암호화
- 계좌 잔고 요청

로그인 화면을 띄우는 코드를 작성했습니다. OnEventConnect 함수를 활용해서 계좌번호를 화면에 바인딩하도록 하겠습니다.

```
public Form1()
{
    InitializeComponent();
    axKHOpenAPI1.CommConnect();
    axKHOpenAPI1.OnEventConnect += onEventConnect;
}
```

OnEventConnect 함수를 Form1.cs의 생성자에 추가하고 함수의 내용을 작성합니다.

```
public void onEventConnect(object sender,
    AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{}
```

OnEventConnect 함수에서 화면에 계좌번호를 바인딩 합니다.

```
public void onEventConnect(object sender ,  
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)  
{  
    if (e.nErrCode == 0)  
    {  
        string accountlist = axKHOpenAPI1.GetLoginInfo("ACCLIST");  
        string[] accountArray = accountlist.Split(':' );  
        for (int i = 0; i < accountArray.Length; i++)  
        {  
            accountComboBox.Items.Add(accountArray[i]);  
        }  
    }  
}
```

- 로그인
- 화면에 계좌번호 정보 바인딩
- 비밀번호 암호화
- 계좌 잔고 요청

다음은 비밀번호를 암호화 하도록 하겠습니다. 화면의 passwordTextBox에 비밀번호를 입력 할 때 비밀번호가 그대로 드러나지 않도록 처리하는 과정입니다.

Form1.cs의 생성자에 함수를 추가합니다.

```
public Form1()  
{  
    InitializeComponent();  
    axKHOpenAPI1.CommConnect();  
    axKHOpenAPI1.OnEventConnect += onEventConnect;  
    passwordTextBox.TextChanged += encryptPassword;  
}
```

passwordTextBox 의 text가 변경되었을 때 동작할 함수를 추가합니다. encryptPassword 함수는 passwordTextBox 의 text가 변경되면 동작합니다.

```
public void encryptPassword(object sender , EventArgs e)  
{  
    if (sender.Equals(passwordTextBox))  
    {  
        passwordTextBox.PasswordChar = '*';  
        passwordTextBox.MaxLength = 4;  
    }  
}
```

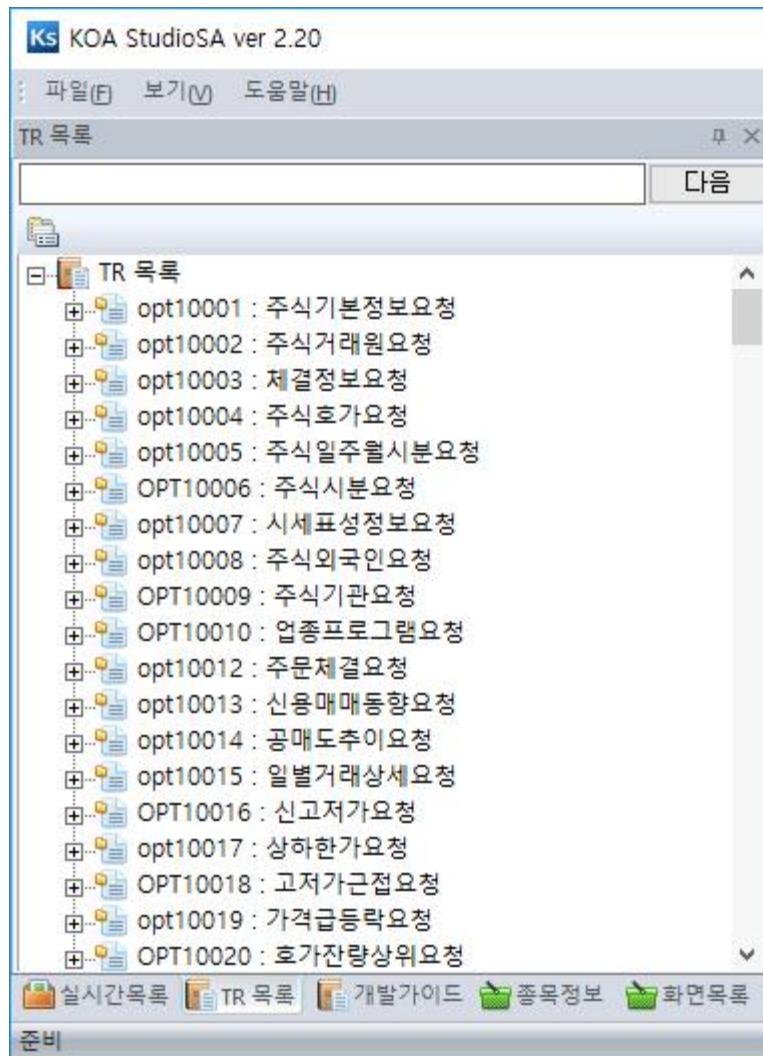
if문을 사용해서 이벤트가 일어난 개체가 passwordTextBox 인지 구분합니다. passwordTextBox 의 문자열은 passwordChar을 사용해서 지정한 문자로 암호화 할 수 있습니다. MaxLength는 문자열의 최대 길이입니다. passwordTextBox에는 계좌의 4자리 비밀번호를 적기 때문에 MaxLength를 4로 지정했습니다.

- 로그인
- 화면에 계좌번호 정보 바인딩
- 비밀번호 암호화
- 계좌 잔고 요청

암호화를 마쳤다면 계좌 잔고 요청 버튼을 클릭했을 때 동작할 코드를 작성합니다.

계좌 잔고 정보를 알아내기 위해서는 KOA Studio의 TR목록을 활용해야 합니다.

tr 목록



opw00018을 클릭하면 아래와 같은 설명이 나타납니다.

[opw00018 : 계좌평가잔고내역요청]

1. Open API 조회 함수 입력값을 설정합니다.

계좌번호 = 전문 조회할 보유계좌번호

```
SetInputValue("계좌번호"      , "입력값 1");
```

비밀번호 = 사용안함(공백)

```
SetInputValue("비밀번호"      , "입력값 2");
```

비밀번호입력매체구분 = 00

```
SetInputValue("비밀번호입력매체구분" , "입력값 3");
```

조회구분 = 1:합산, 2:개별

```
SetInputValue("조회구분"      , "입력값 4");
```

2. Open API 조회 함수를 호출해서 전문을 서버로 전송합니다.

TR 목록 opw00018의 설명대로 코드를 작성합니다. 먼저 버튼이 클릭되었을 때 동작할 함수 ButtonClicked를 Form1에 추가합니다. ButtonClicked 함수에서 opw00018 요청을 보내는 코드를 작성 할 것입니다.

```
public Form1()
{
    InitializeComponent();
    axKHOpenAPI1.CommConnect();
    axKHOpenAPI1.OnEventConnect += onEventConnect;
    passwordTextBox.TextChanged += encryptPassword;
    balanceCheckButton.Click += ButtonClicked;
}
```

함수의 내용을 작성합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
}
```

if문을 사용해서 요청 객체를 구분하고 함수의 내용을 작성합니다. 화면에 있는 계좌번호, 비밀번호 정보를 활용합니다. ButtonClicked 함수에서 CommRqData요청을 보냅니다. KOA Studio의 설명처럼 SetInputValue부터 차례대로 작성합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
    if (sender.Equals(balanceCheckButton))
    {
        if (accountComboBox.Text.Length > 0 && passwordTextBox.Text.Length > 0)
        {
            string accountNumber = accountComboBox.Text;
            string password = passwordTextBox.Text;

            axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
            axKHOpenAPI1.SetInputValue("비밀번호", password);
            axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
            axKHOpenAPI1.SetInputValue("조회구분", "1");

            axKHOpenAPI1.CommRqData("계좌평잔고내역요청",
                "opw00018", 0, "8100");
        }
    }
}
```

프로그램을 실행하고 accountComboBox에서 선택한 문자열은 accountComboBox.Text를 사용해서 가져올 수 있습니다. passwordTextBox에 작성한 비밀번호도 passwordTextBox.Text를 사용해서 가져옵니다.

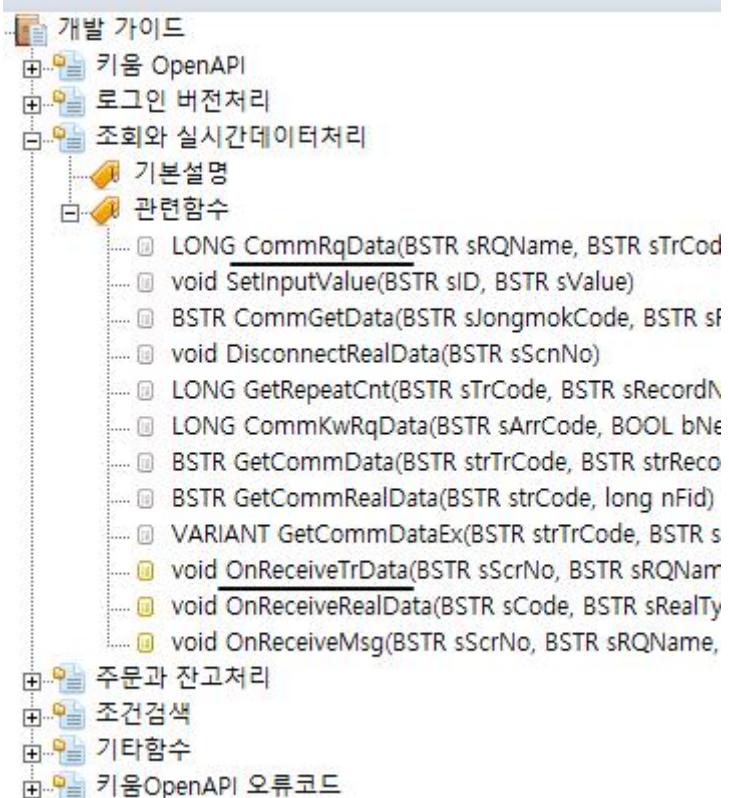
이제 Form1.cs의 생성자에 OnReceiveTrData 함수를 추가하고 작성합니다. 원하는 정보를 획득하는 과정은 아래와 같습니다.

- CommRqData 함수 호출
- OnReceiveTrData 이벤트 함수 호출
- OnReceiveTrData 이벤트 함수에서 GetCommData 함수 호출
- 계좌 잔고 정보 획득

```
public Form1()
{
    InitializeComponent();
    axKHOpenAPI1.CommConnect();
    axKHOpenAPI1.OnEventConnect += onEventConnect;
    passwordTextBox.TextChanged += encryptPassword;
    balanceCheckButton.Click += ButtonClicked;
    axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
}
```

추가한 `onReceiveTrData` 함수의 내용을 작성합니다. `onReceiveTrData` 함수는 아래 그림의 `CommRqData` 함수를 호출하면 호출되는 이벤트 함수입니다. 사용자가 원하는 동작을 수행 하도록 호출될 `OnReceiveTrData` 함수의 내용을 작성합니다.

`axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;` 를 Form1의 생성자에 작성합니다. 사용자가 작성하는 함수를 `OnReceiveTrData` 함수에 덮어쓰기 하는 과정입니다.



`OnReceiveTrData` 함수의 두번째 입력 변수의 타입은 `AxKHOpenAPILib.DKHOpenAPIEventsOnReceiveTrDataEvent`입니다. 이벤트 타입의 변수 `e`에는 사용자가 `CommRqData` 함수를 사용해서 보냈던 요청 사항의 정보가 담겨 있습니다. 이벤트 함수에서는 일반함수로 요청했던 사항에 대해 더 구체적인 요청을 보낼 수 있습니다. 이벤트 타입의 변수 `e`에 어떤 정보들이 담겨 있는지 아래 설명을 통해서 살펴보도록 하겠습니다.

OnReceiveTrData 함수의 설명은 아래와 같습니다.

[OnReceiveTrData() 이벤트]

```
void OnReceiveTrData(
    BSTR sScrNo,          // 화면번호
    BSTR sRQName,         // 사용자 구분명
    BSTR sTrCode,          // TR이름
    BSTR sRecordName,      // 레코드 이름
    BSTR sPrevNext,        // 연속조회 유무를 판단하는 값 0: 연속(추가조회)데이터 없음, 1:
                           // 연속(추가조회) 데이터 있음
    LONG nDataLength,      // 사용안함.
    BSTR sErrorCode,       // 사용안함.
    BSTR sMessage,          // 사용안함.
    BSTR sSplmMsg           // 사용안함.
)
```

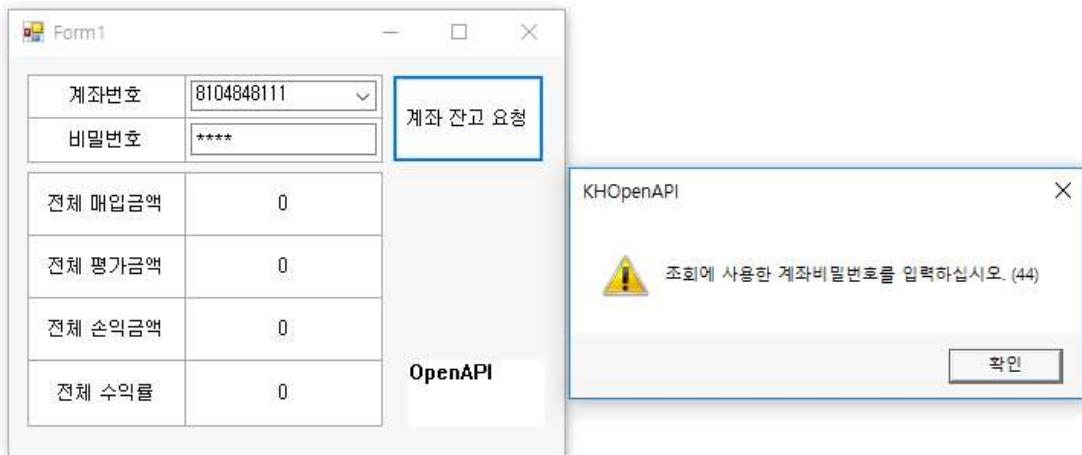
조회요청 응답을 받거나 조회데이터를 수신했을때 호출됩니다.

조회데이터는 이 이벤트 함수내부에서 GetCommData()함수를 이용해서 얻어올 수 있습니다.

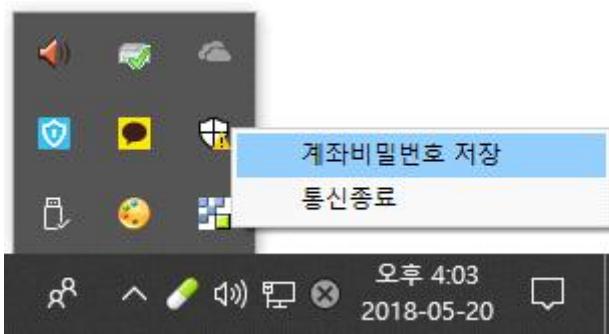
변수 e를 사용해서 화면번호 , 사용자구분명 , 요청했던 TR이름 , 레코드이름 , 연속조회 유무를 알 수 있습니다.

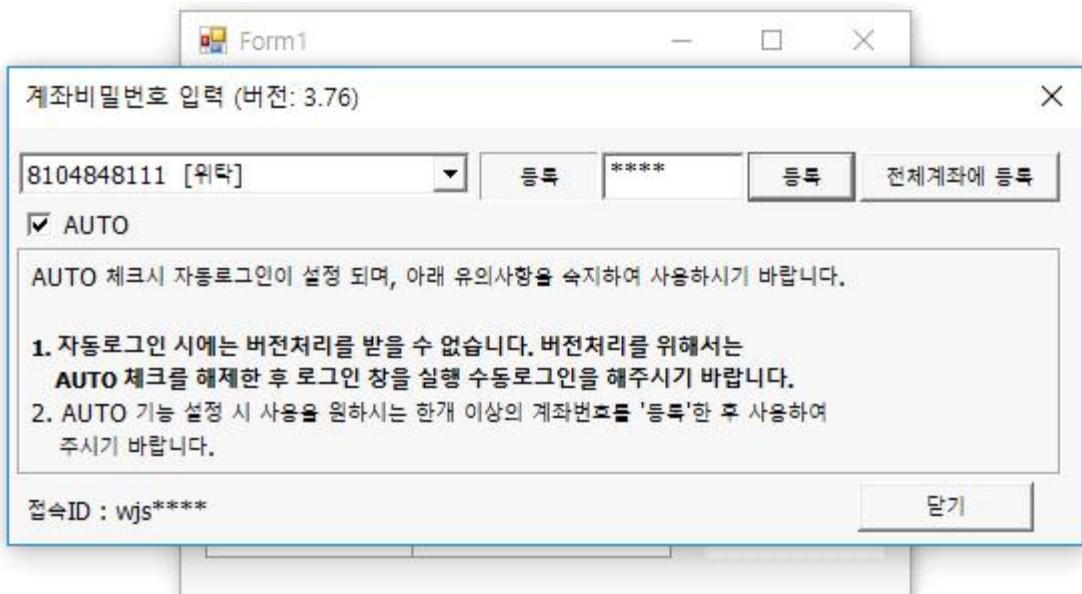
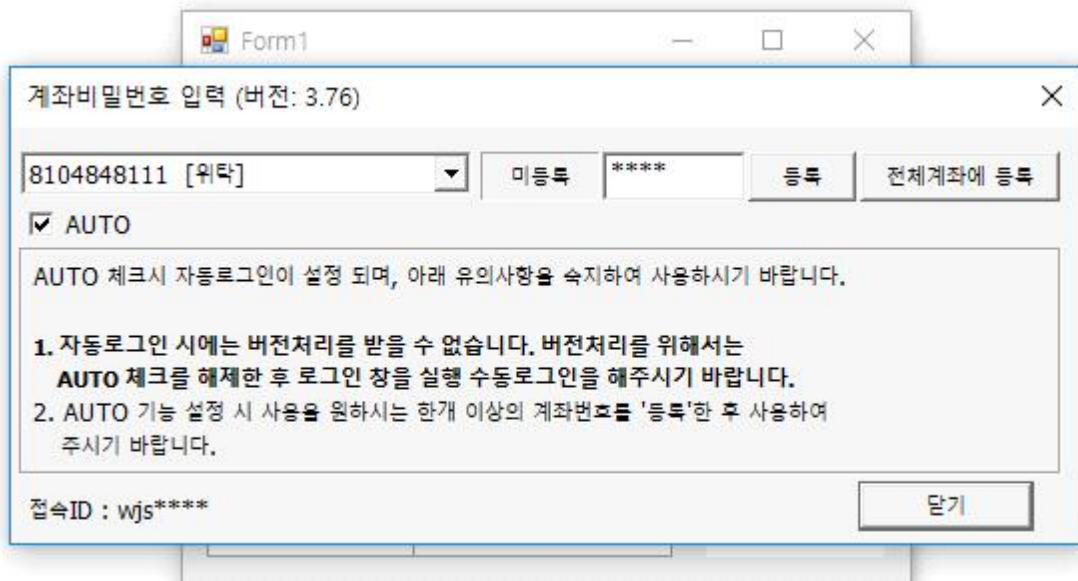
```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    Console.WriteLine(e.sScrNo);
    Console.WriteLine(e.sRQName);
    Console.WriteLine(e.sTrCode);
    Console.WriteLine(e.sRecordName);
    Console.WriteLine(e.sPrevNext);
}
```

함수 안에서 Console.WriteLine을 사용해서 콘솔에 정보들이 출력되도록 할 수 있습니다. 여기까지 코드를 작성하고 실행했을 때 아래 그림과 같은 에러가 나타날 수도 있습니다.



이때 확인 버튼을 클릭하고 컴퓨터 화면 오른쪽 하단의 아래 그림 아이콘을 클릭하고 비밀 번호를 저장해줍니다. AUTO를 체크하면 앞으로 자동으로 비밀번호가 입력됩니다.





닫기를 클릭하고 다시 비밀번호를 입력해서 요청하면 프로그램이 잘 동작합니다.

콘솔 창에 아래 항목들이 출력 됩니다.

```
8100
계좌평가잔고내역요청
opw00018
```

요청했던 화면번호 , 사용자구분명 , TR이름이 출력되고 있습니다. 현재 요청에서는 레코드 이름 , 연속 조회여부는 없습니다.

onReceiveTrData 함수에서 사용자 구분명을 사용해서 요청을 구분하고 구체적인 데이터를 요청해보도록 하겠습니다. 데이터 요청은 GetCommData 함수를 사용해서 진행합니다. GetCommData 함수에 대한 설명은 아래와 같습니다. 아래 설명은 GetCommData 함수를 호출할때 입력변수를 어떻게 작성해야 하는지 말해줍니다.

[GetCommData() 함수]

```
GetCommData(
BSTR strTrCode,    // TR 이름
BSTR strRecordName, // 레코드이름
long nIndex,        // TR반복부
BSTR strItemName) // TR에서 얻어오려는 출력항목이름
```

OnReceiveTRData()이벤트 함수가 호출될때 조회데이터를 얻어오는 함수입니다.

이 함수는 반드시 OnReceiveTRData()이벤트 함수가 호출될때 그 안에서 사용해야 합니다.

OnReceiveTRData()이벤트 함수가 호출될때 조회데이터를 얻어오는 함수입니다. 이 함수는 반드시 OnReceiveTRData()이벤트 함수가 호출될때 그 안에서 사용해야 합니다.

여기서는 2번째 입력변수가 레코드 이름입니다. 하지만 함수를 호출 할때는 요청 이름을 적으면 됩니다. 아래 예시처럼 코드를 작성하게 됩니다.

```
GetCommData(Tr이름, 요청이름 , TR반복 횟수 , TR목록의 output 에서 얻고자 하는 항목의 이름)
```

함수의 괄호 안에 입력변수를 모두 입력하면 아래 코드처럼 됩니다.

```
axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "총매입금액")
```

아래 코드를 통해서 더 자세히 살펴보도록 하겠습니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName=="계좌평가잔고내역요청")
    {
        long totalPurchase = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총매입금액"));
    }
}
```

이벤트 변수 e에는 이미 TR이름, 사용자 요청이름이 들어있습니다. 그래서 e.sTrCode, e.sRQName을 적습니다. 반복횟수는 없으므로 0을 적습니다. TR에서 얻어오려는 항목의 이름은 "총매입금액"을 적었습니다.

long.Parse는 Parse() 괄호 안의 문자열에 들어있는 숫자를 추출하는 명령어입니다. 예를 들면 "123"을 숫자 123으로 바꿀 수 있습니다.

나머지 다른 항목들도 요청하고 화면의 Label들에 바인딩 해보도록 하겠습니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName=="계좌평가잔고내역요청")
    {
        long totalPurchase = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총매입금액"));

        long totalEstimate = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총평가금액"));

        long totalProfitLoss = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총평가손익금액"));

        double totalProfitRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총수익률(%")));

        totalProfitRateLabel.Text = String.Format("{0:#,###}",totalPurchase);
        totalEstimateLabel.Text = String.Format("{0:#,###}", totalEstimate);
        totalProfitLabel.Text = String.Format("{0:#,###}", totalProfitLoss);
        totalProfitRateLabel.Text = String.Format("{0:f2}",totalProfitRate);
    }
}
```

매입금액 , 평가금액 , 손익금액은 모두 long.Parse를 사용해서 추출했습니다. 이렇게 하는 이유는 String.Format 함수를 사용하기 위해서입니다. String.Format을 사용하면 원하는 형태로 화면에 바인딩 할 수 있습니다. 예를 들어서 현재 작성된 코드처럼 숫자의 3자리마다 쉼표를 찍도록 코드를 작성 할 수 있습니다. **123,456,789** 3자리마다 쉼표를 찍기 위해서는 String.Format("{0:#,###}")를 사용하면 됩니다. String.Format("{0:f2}")는 소수점 둘째 자리까지 소수를 표기하기 위한 방법입니다.

4. 종목 검색

Preview

4장에서는 주식 종목을 검색해봅니다. 종목의 가격 정보를 화면에 출력해서 확인합니다. 실시간으로 변화하는 주식의 가격을 화면에서 확인합니다. 4장의 내용은 종목 데이터를 프로그램에서 확인하는데 도움이 됩니다.

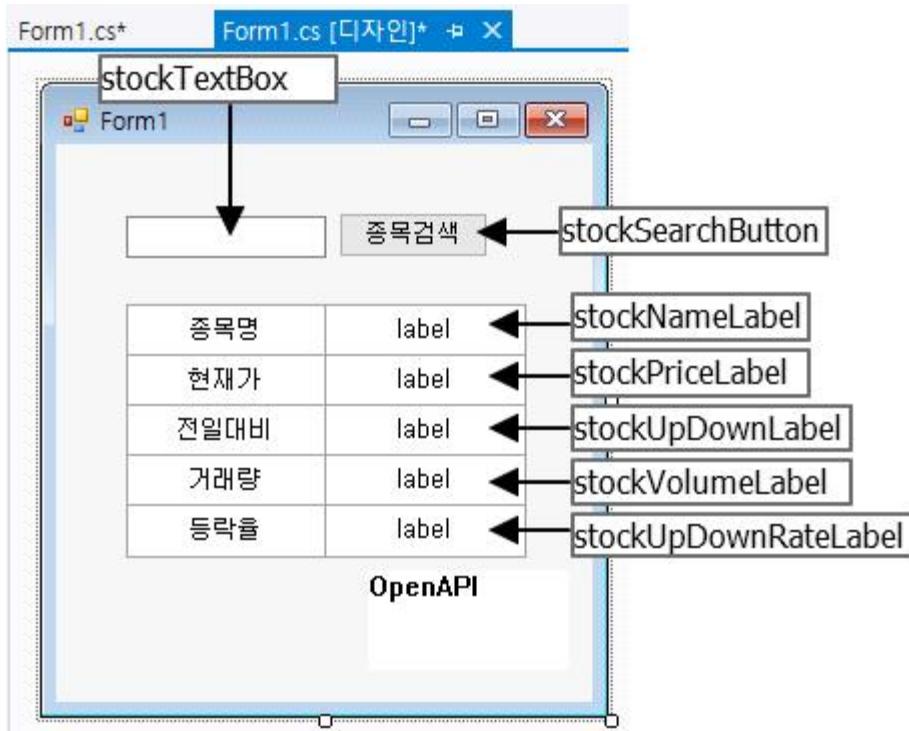
Contents

4.1장에서는 종목 정보를 조회합니다. CommRqData 함수와 OnReceiveTrData 함수를 사용해서 종목 정보를 확인합니다. List를 사용해서 종목 데이터를 다루는 방법을 알아봅니다.

4.2장에서는 실시간으로 변화하는 종목 정보를 조회해봅니다. CommRqData 함수와 OnReceiveRealData 함수를 사용해서 종목 정보를 실시간으로 확인합니다.

4.1 종목 조회

많은 사람들은 hts에서 종목을 주문하기 전에 종목을 검색 해봅니다. 4.1장에서는 주식 종목의 정보를 요청하고 프로그램에서 확인하는 방법에 대해서 알아보겠습니다. 먼저 화면을 구성합니다



- 테이블 레이아웃 패널 생성
- stockTextBox 텍스트 박스 생성
- stockSearchButton 검색버튼 생성
- 화면 레이블 생성

stockTextBox의 속성

속성	값
(Name)	stockTextBox
AutoCompleteMode	Suggest
AutoCompleteSource	CustomSource

stockSearchButton의 속성

속성	값
(Name)	stockSearch
Text	종목검색

stockNameLabel의 속성

속성	값
(Name)	stockNameLabel
Anchor	Top, Bottom, Left, Right
Text	label

stockPriceLabel의 속성

속성	값
(Name)	stockPriceLabel
Anchor	Top, Bottom, Left, Right
Text	label
.TextAlign	MiddleCenter

stockUpDownLabel의 속성

속성	값
(Name)	stockUpDownLabel
Anchor	Top, Bottom, Left, Right
Text	label
.TextAlign	MiddleCenter

stockVolumeLabel의 속성

속성	값
(Name)	stockVolumeLabel
Anchor	Top, Bottom, Left, Right
Text	label
.TextAlign	MiddleCenter

stockUpDownRateLabel의 속성

속성	값
(Name)	stockUpDownRateLabel
Anchor	Top, Bottom, Left, Right
Text	label
.TextAlign	MiddleCenter

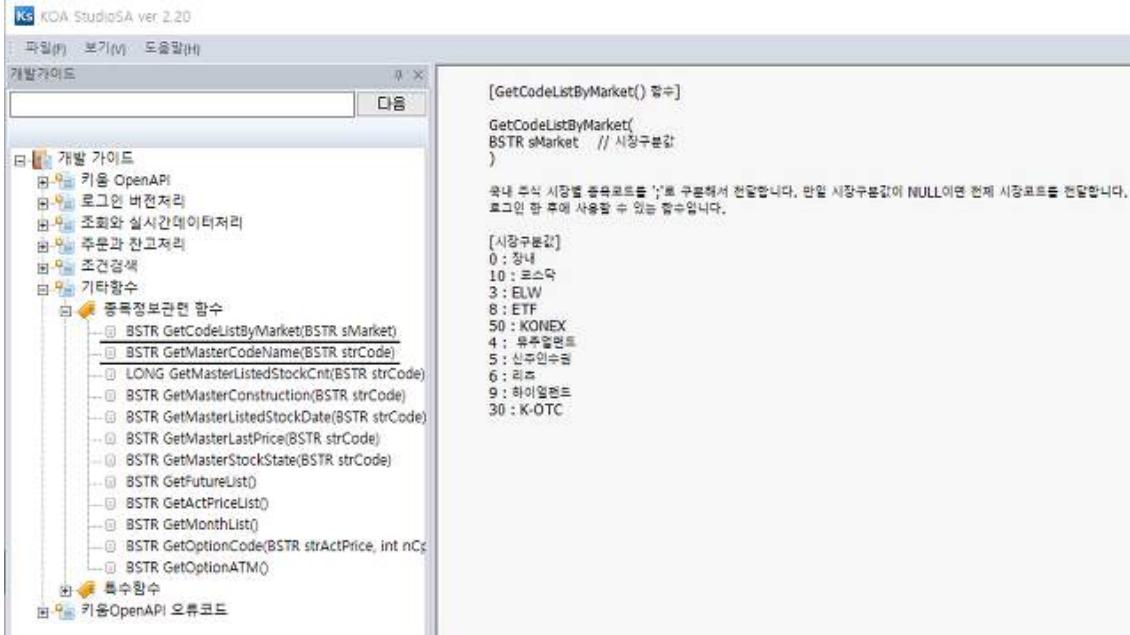
화면을 모두 구성했다면 3.1장에서 했던 것처럼 Form1.cs의 생성자에 로그인 코드를 먼저 작성합니다.

```
axKHOpenAPI1.GetCodeListByMarket(null);
```

이번 장에서는 종목의 이름을 검색해서, 종목의 가격 정보를 알아보도록 하겠습니다. 이번 장에서는 아래와 같은 순서로 코드를 작성하도록 하겠습니다.

- 종목 코드 목록 가져오기 □
- 종목 코드 목록으로 종목 이름 목록 알아내기 □
- 종목 이름 목록을 화면에 추가하기 □

종목 코드 목록을 요청하는 함수는 GetCodeListByMarket() 입니다.



아래와 같이 코드를 작성해서 요청하면 됩니다.

```
axKHOpenAPI1.GetCodeListByMarket(null);
```

GetCodeListByMarket() 함수는 전달하는 인자값에 따라서 장내, 코스닥, ELW 등의 시장을 구분하고 종목 코드를 요청 할 수 있습니다. null을 함수의 인자로 전달하면 주식시장에 있는 모든 종목의 코드를 요청합니다. CommConnect 함수로 로그인하면 호출 되는 OnEventConnect 이벤트 함수에 코드를 작성하도록 하겠습니다.

```
public void onEventConnect(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    if (e.nErrCode==0)
    {
        string stockCodeList = axKHOpenAPI1.GetCodeListByMarket("0");
        string[] stockCode = stockCodeList.Split(';');
    }
}
```

이렇게 코드를 작성하면 종목코드를 요청하고 배열에 저장할 수 있습니다. 이제 종목 코드들과 종목 코드들에 해당하는 이름을 프로그램이 실행되는 동안 컴퓨터에 저장해보도록 하겠습니다. 키움 API서버에 종목 정보를 요청하기 위해서는 종목 코드를 알아야 합니다. 종목 코드들과 종목 코드들에 해당하는 이름들을 컴퓨터에 저장하면 사용자가 종목 코드를 몰라도 종목 이름 만으로 종목 정보를 조회 할 수 있도록 코드를 작성 할 수 있습니다.

종목의 이름은 종목코드와 GetMasterCodeName() 함수를 사용해서 요청 할 수 있습니다.

요청의 결과로 반환될 종목들을 하나의 객체로 보고 코드를 작성해보도록 하겠습니다. 주식 종목 1개는 종목코드와 종목 이름을 갖습니다. string 타입의 변수 stockCode , stockName을 갖는 stockInfo 클래스를 하나 선언하는 것으로 주식 종목을 모델링 합니다.

```
class stockInfo
{
    public string stockCode;
    public string stockName;
    public stockInfo(string code, string name)
    {
        this.stockCode = code;
        this.stockName = name;
    }
}
```

Form1 클래스에 종목코드와 종목이름 정보를 담을 리스트를 선언합니다.

```
List<stockInfo> stockList;
```

로그인해서 OnEventConnect 함수가 호출 될 때 리스트를 초기화합니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    stockList = new List<stockInfo>();

    if (e.nErrCode == 0)
    {
        AutoCompleteStringCollection stockcollection = new
AutoCompleteStringCollection();

        string stockCode = axKHOpenAPI1.GetCodeListByMarket(null);
        string[] stockCodeArray = stockCode.Split(':');
        for (int i = 0; i < stockCodeArray.Length; i++)
        {
            stockList.Add(new stockInfo(stockCodeArray[i],
axKHOpenAPI1.GetMasterCodeName(stockCodeArray[i])));
        }
        for (int i = 0; i < stockList.Count; i++)
        {
            stockcollection.Add(stockList[i].stockName);
        }
        stockTextBox.AutoCompleteCustomSource = stockcollection;
    }
}
```

리스트를 초기화하고 종목코드, 종목코드이름을 리스트에 추가하고 있습니다. GetMasterCodeName()의 괄호 안에 종목 코드를 입력해서 요청을 보내면 종목 코드에 해당하는 종목 이름을 반환해줍니다. AutoCompleteStringCollection은 textbox에서 사용할 자동완성 기능의 기반이 되는 목록입니다. collection을 구성하고 TextBox의 AutoCompleteCustomSource에 추가합니다.

종목을 자동완성에 추가했다면 이제 종목검색 버튼을 클릭했을때 종목정보를 종목 정보를 요청하도록 해야 합니다.

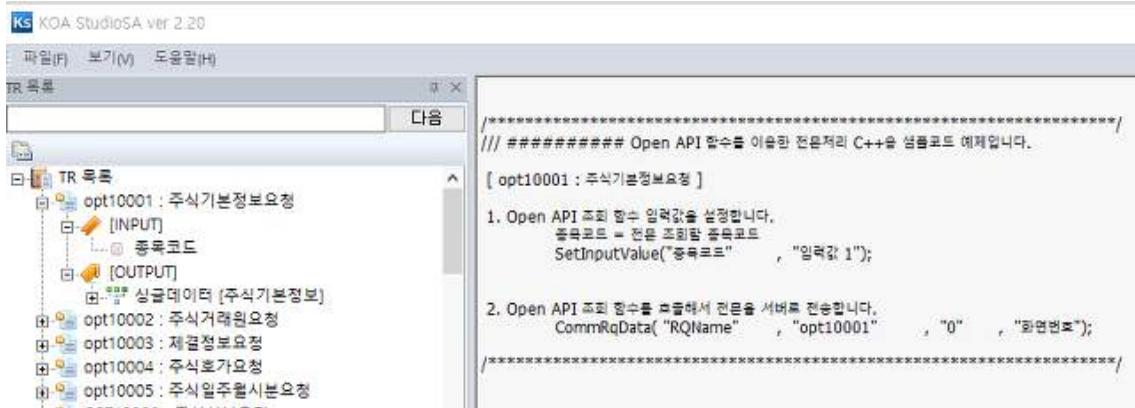
```
searchButton.Click += stockSearch;
```

searchButton.Click에 함수를 추가합니다. stockSearch 함수에서 CommRqData 함수를 통해서 종목의 정보를 요청 할 것입니다.

```
public void stockSearch(object sender,EventArgs e)
{
    string searchStock = stockTextBox.Text;
    int index = stockNameList.FindIndex(o=>o==searchStock);
    string stockCode = stocklist[index];
    axKHOpenAPI1.SetInputValue("종목코드", stockCode);
    axKHOpenAPI1.CommRqData("종목정보요청","opt10001",0,"5000");
}
```

TextBox로부터 입력한 종목이름을 받아오고, 리스트에서 입력한 종목 이름에 해당하는 인덱스가 몇 번인지 찾습니다. 찾은 인덱스로 종목 코드가 있는 리스트에서 종목 코드를 찾습니다. 찾은 종목 코드를 사용해서 종목 정보를 요청합니다.

CommRqData 함수는 OnReceiveTrData 함수를 호출 합니다. CommRqData로 요청을 보내는 방법은 KOA Studio를 참고하시면 됩니다.



`SetInputValue`에 종목코드를 입력합니다. `CommRqData` 함수에는 사용자 정의 요청 이름, TR 목록 번호, 연속조회여부, 화면번호를 입력합니다.

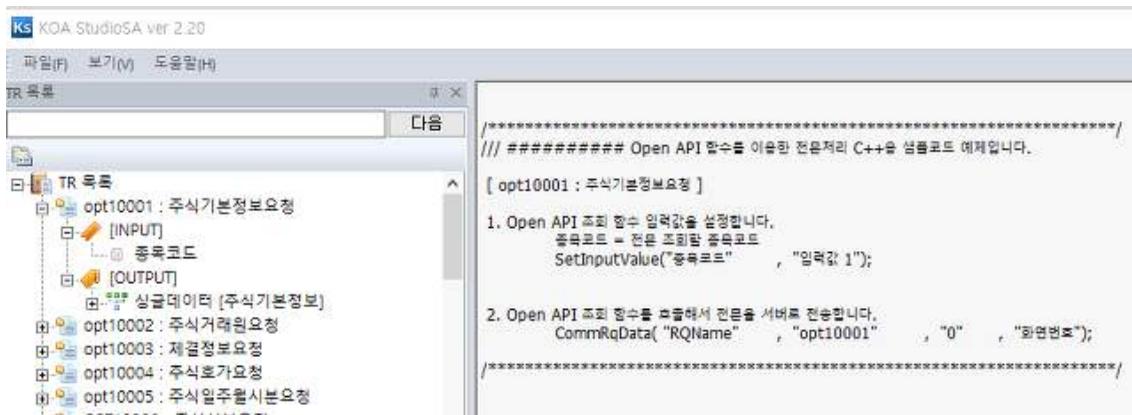
OnReceiveTrData 함수를 사용해서 종목의 현재가격, 거래량, 등락률 등의 정보를 호출해 보도록하겠습니다. Form1.cs의 생성자에 OnReceiveTrData 함수를 추가합니다.

```
axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
```

추가한 onReceiveTrData 함수의 내용을 작성합니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName == "종목정보요청")
    {
        // ...
    }
}
```

onReceiveTrData 함수의 OnReceiveTrDataEvent 타입 매개변수 e에는 사용자가 보낸 요청에 대한 정보가 담겨 있습니다.



e.sRQName에는 사용자가 CommRqData 함수를 호출 할 때 지정한 요청 이름이 담겨 있습니다.

e.sRQName에 담긴 값이 CommRqData에 작성했던 "종목정보요청"이 맞다면 GetCommData함수를 호출해서 주식 종목 정보를 요청합니다.

TR 목록 opt10001의 output에 적힌 글자를 입력해서 함수를 호출하면 됩니다.

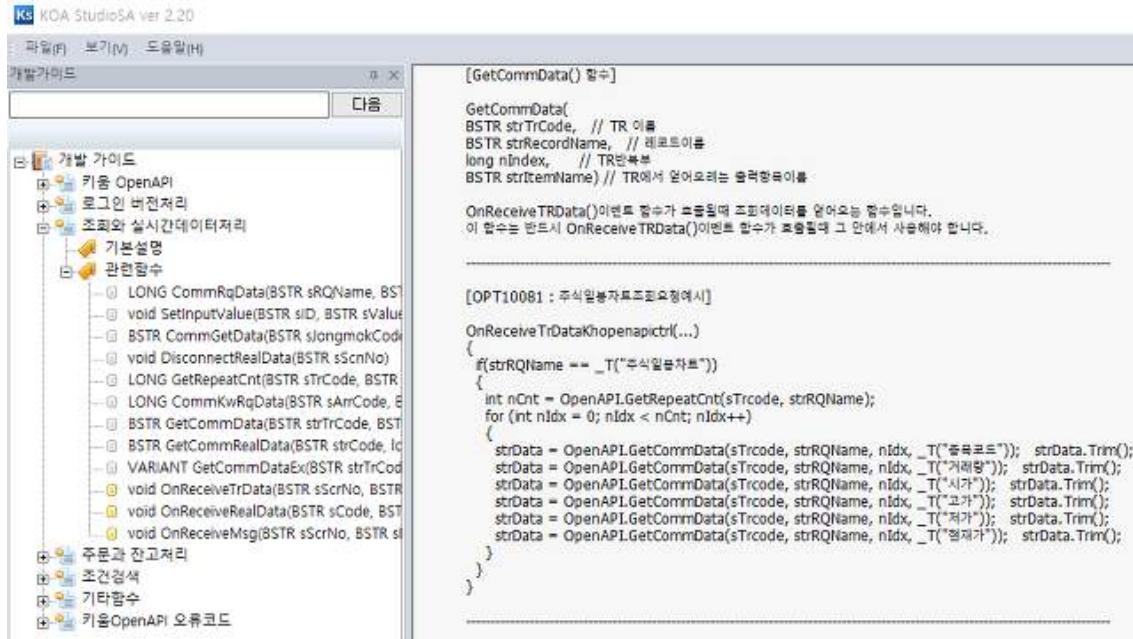
```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName == "종목정보요청")
    {
        long stockPrice=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가").Trim().Replace("-", ""));
        string stockName=axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "종
목명").Trim();
        long upDown=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "전일대비").Trim());
        long volume=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "거래량").Trim());
        string upDownRate=axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "등락율").Trim();

        stockPriceLabel.Text=String.Format("{0:#,###}", stockPrice);
    }
}

```

getcomm



KOA Studio처럼 e.Tr코드 , e.사용자요청이름, 연속요청여부,output이름을 함수 호출시에 작성합니다.

코드에서 함수 호출 이후에 Trim()을 붙인 이유는 opt10001 현재가 , 거래량 , 요청등의 경우 문자열 앞에 공백이 함께 반환됩니다. 그래서 공백을 제거하기 위해서 문자열 뒤에 Trim() 함수를 호출해줍니다. 그리고 키움증권에서는 현재 가격이 전일보다 올랐으면 + 기호, 내렸으면 - 기호가 포함되어서 사용자에게 반환됩니다. 이 -와 +기호를 subString 함수를 사용해서 제거해줍니다. 공백을 제거한 후에 숫자 3자리마다 쉼표를 찍기 위해서 String.Format 함수를 사용합니다. String.Format 함수를 사용하려면 먼저 int.Parse를 사용해서 숫자가 들어있는 문자열을 int 형으로 변환합니다.

```
String.Format("{0:#,###}", stockPrice)
```

int 형으로 변환한 값을 String.Format에 넣고 "0:#,###"도 함께 입력합니다. 3자리마다 쉼표를 표시하라는 의미입니다.

값을 잘 변환했다면 이제 화면에 바인딩합니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName == "종목정보요청")
    {
        long stockPrice=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가").Trim().Replace("-", ""));
        string stockName=axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "종
목명").Trim();
        long upDown=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "전일대비").Trim());
        long volume=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "거래량").Trim());
        string upDownRate=axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "
등락율").Trim();

        stockPriceLabel.Text=String.Format("{0:#,###}", stockPrice);
        stockNameLabel.Text=stockName;
        stockUpDownLabel.Text = String.Format("{0:#,###}", upDown);
        stockVolumeLabel.Text = String.Format("{0:#,###}", volume);
        if (upDown ==0)
        {
            stockUpDownLabel.Text = "0";
        }
        if (volume == 0)
        {
            stockVolumeLabel.Text = "0";
        }
        stockUpDownRateLabel.Text = upDownRate+"%";
```

```

    }
}

```

label.Text에 값을 저장합니다.

아래 코드는 Form1.cs에 작성한 전체 코드입니다.

전체 코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp3
{
    public partial class Form1 : Form
    {
        List<stockInfo> stockList;

        public Form1()
        {
            InitializeComponent();
            axKHOpenAPI1.CommConnect();
            axKHOpenAPI1.OnEventConnect += onEventConnect;
            axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
            stockSearchButton.Click += stockSearch;
        }

        public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
        {
            if (e.sRQName=="종목정보요청")
            {
                l          o          n          g
                stockPrice=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,e.sRQName,0,"현재가"
").Trim().Replace("-", ""));
                string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "종목명").Trim();
                long upDown = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "전일대비").Trim());
                long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,

```

```

e.sRQName, 0, "거래량").Trim());
    string      upDownRate      =      axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "등락율").Trim();

    stockPriceLabel.Text = String.Format("{0:#,###}", stockPrice);
    stockNameLabel.Text = stockName;
    stockUpDownLabel.Text = String.Format("{0:#,###}", upDown);
    stockVolumeLabel.Text = String.Format("{0:#,###}", volume);
    if (upDown ==0)
    {
        stockUpDownLabel.Text = "0";
    }
    if (volume == 0)
    {
        stockVolumeLabel.Text = "0";
    }
    stockUpDownRateLabel.Text = upDownRate+"%";
}
}

public void stockSearch(object sender , EventArgs e)
{

    string stockName = stockTextBox.Text;
    int index = stockList.FindIndex(o => o.stockName == stockName);
    string stockCode = stockList[index].stockCode;
    currentStockCode = stockCode;

    axKHOpenAPI1.SetInputValue("종목코드",stockCode);
    axKHOpenAPI1.CommRqData("종목정보요청","opt10001",0,"5000");
}
public      void      onReceiveRealData(object      sender      ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    Console.WriteLine(e.sRealType);

}
public      void      onEventConnect(object      sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    stockList = new List<stockInfo>();
}

```

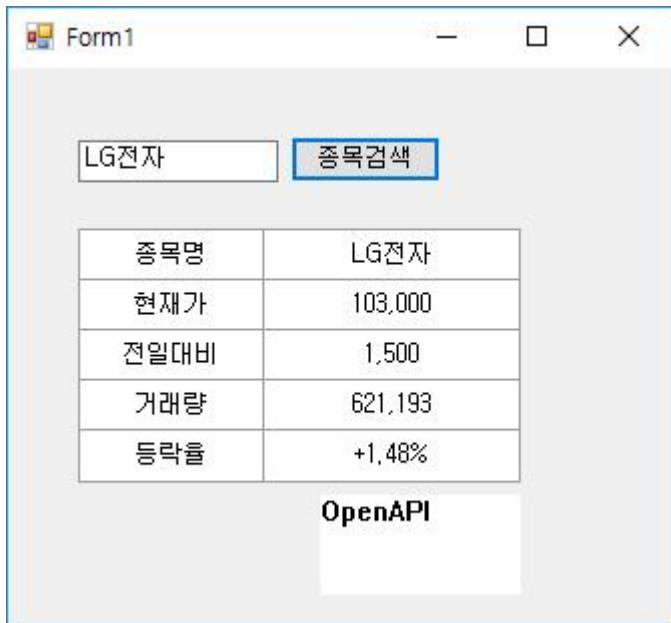
```
if (e.nErrCode==0)
{
    AutoCompleteStringCollection stockcollection = new
AutoCompleteStringCollection();

    string stockCode= axKHOpenAPI1.GetCodeListByMarket(null);
    string [] stockCodeArray = stockCode.Split('\'');
    for (int i = 0; i < stockCodeArray.Length; i++)
    {
        stockList.Add(new stockInfo(stockCodeArray[i],
axKHOpenAPI1.GetMasterCodeName(stockCodeArray[i])));
    }
    for (int i = 0; i < stockList.Count; i++)
    {
        stockcollection.Add(stockList[i].stockName);
    }
    stockTextBox.AutoCompleteCustomSource = stockcollection;
}

}
}

class stockInfo
{
    public string stockCode;
    public string stockName;
    public stockInfo(string code,string name)
    {
        this.stockCode = code;
        this.stockName = name;
    }
}
```

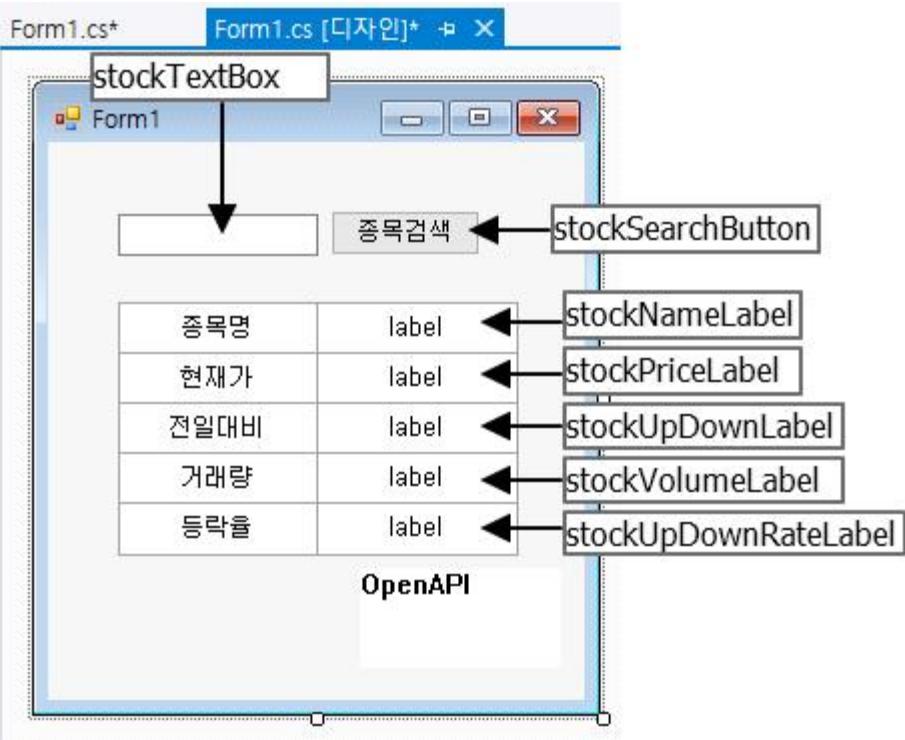
실행화면입니다.



4.2장에서는 실시간으로 가격 정보가 바뀌는 것을 프로그램에 반영해보도록 하겠습니다.

4.2 실시간 종목 조회

지속적으로 변화하는 주식 시장에서 실시간으로 종목 정보를 알아내는 것은 중요합니다. 4.2장에서는 주식 종목의 정보를 요청하고 실시간으로 프로그램에서 확인하는 방법에 대해서 알아보겠습니다. 먼저 화면을 구성합니다.



- 테이블 레이아웃 패널 생성
- stockTextBox 텍스트 박스 생성
- stockSearchButton 검색버튼 생성
- 화면 레이블 생성

stockTextBox의 속성

속성	값
(Name)	stockTextBox
AutoCompleteMode	Suggest
AutoCompleteSource	CustomSource

stockSearchButton의 속성

속성	값
(Name)	stockSearchButton
Text	종목검색

stockNameLabel의 속성

속성	값
(Name)	stockNameLabel
Anchor	Top, Bottom, Left, Right
Text	label
TextAlign	MiddleCenter

stockPriceLabel의 속성

속성	값
(Name)	stockPriceLabel
Anchor	Top, Bottom, Left, Right
Text	label
TextAlign	MiddleCenter

stockUpDownLabel의 속성

속성	값
(Name)	stockUpDownLabel
Anchor	Top, Bottom, Left, Right
Text	label
TextAlign	MiddleCenter

stockVolumeLabel의 속성

속성	값
(Name)	stockVolumeLabel
Anchor	Top, Bottom, Left, Right
Text	label
TextAlign	MiddleCenter

stockUpDownRateLabel의 속성

속성	값
(Name)	stockUpDownRateLabel
Anchor	Top, Bottom, Left, Right
Text	label
TextAlign	MiddleCenter

화면을 모두 구성했다면 4.1절에서 했던 것처럼 코드를 먼저 작성합니다. 4.1의 전체 코드를 참고하시면 됩니다.

4.2장에서는 4.1장에서 바인딩 한 가격 정보를 실시간으로 변경 되도록 합니다.

일시적으로 데이터를 조회 할때는 OnReceiveTrData 함수를 사용하지만, 실시간으로 데이터를 조회하려면 OnReceiveRealData를 사용합니다. 4.1에 코드에서 OnReceiveRealData 함수를 추가합니다.

OnReceiveRealData 함수를 Form1.cs의 생성자에 추가하도록 하겠습니다.

```
axKHOpenAPI1.OnReceiveRealData += onReceiveRealData;
```

OnReceiveRealData 함수에서 종목 코드를 사용 할 수 있도록 Form1 클래스에 변수를 선언합니다.

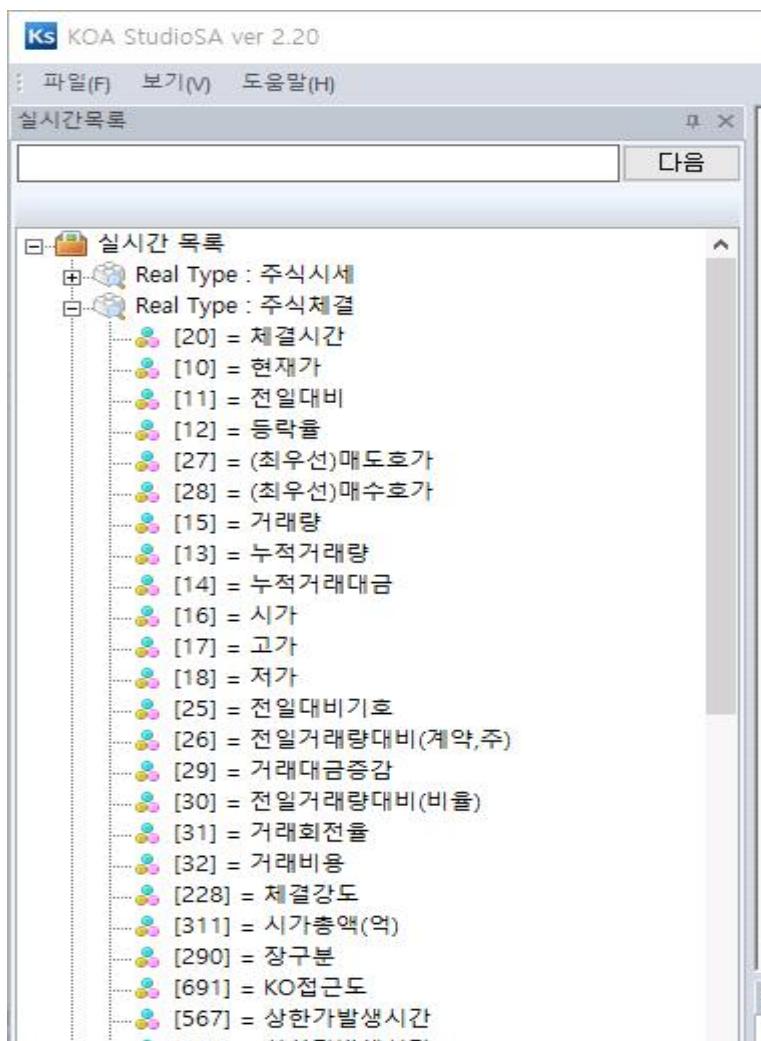
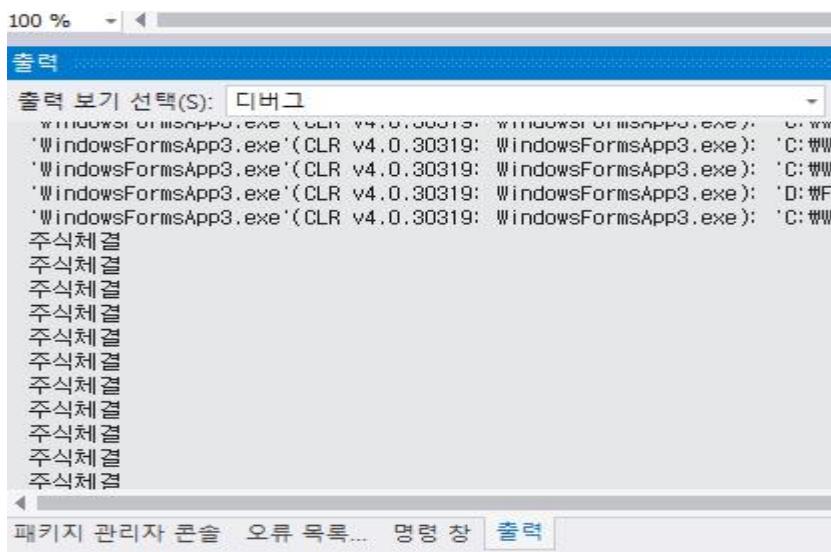
```
public partial class Form1 : Form
{
    List<stockInfo> stockList;
    public string currentStockCode = "";
    public Form1()
    {
        InitializeComponent();
        axKHOpenAPI1.CommConnect();
        axKHOpenAPI1.OnEventConnect += onEventConnect;
        axKHOpenAPI1.OnReceiveRealData += onReceiveRealData;
        axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
        stockSearchButton.Click += stockSearch;
    }
}
```

OnReceiveRealData 함수도 OnReceiveTrData 함수처럼 Event 타입의 변수 e에 요청에 대한 정보가 담겨 있습니다.

```
public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    Console.WriteLine(e.sRealType);
}
```

OnReceiveRealData 함수에서는 e.sRealType에 주식시세, 주식체결, 주식우선호가 등의 값이 담겨서 사용자에게 전달됩니다. OnReceiveRealData 함수는 CommRqData로 사용자가 데이터를 요청하면 호출됩니다.

프로그램을 실행하고, 4.1절 처럼 주식 종목을 검색하면 OnReceiveRealData 함수가 호출되어서 주식체결 같은 RealType이 출력됩니다.



OnReceiveRealType 함수 내에서 GetCommRealData 함수를 호출하면 실시간 데이터를 요청 할 수 있습니다. 출력 창에 나타났던 "주식체결" sRealType을 사용해서 OnReceiveRealData 함수를 작성해보도록 하겠습니다.

```
public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    if (e.sRealType=="주식체결")
    {
        if (e.sRealKey == currentStockCode)
        {

        }
    }
}
```

e.sRealType이 "주식체결"이고 e.sRealKey가 currentStockCode와 동일하다면 현재 조회한 종목의 실시간 정보를 요청합니다.

```
public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    if (e.sRealType=="주식체결")
    {
        if (e.sRealKey == currentStockCode)
        {
            long stockPrice = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode, 10));
            stockPriceLabel.Text = String.Format("{0:#,###}", stockPrice);
        }
    }
}
```

KOASTudio 의 개발가이드에서 GetCommRealData 함수 사용 방법을 참고해서 함수를 작성하면 됩니다. GetCommRealData함수에 종목코드, 실시간Fid를 인자로 적어서 데이터를 요청합니다. Fid 10번은 현재가 입니다. 11. 전일대비 , 12. 등락률 , 15. 거래량 을 추가 요청합니다.

```

public void onReceiveRealData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    if (e.sRealType=="주식체결")
    {
        if (e.sRealKey == currentStockCode)
        {
            long stockPrice = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode, 10));
            long upDown = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode,11));
            string upDownRate = axKHOpenAPI1.GetCommRealData(currentStockCode,12);
            long volume = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode,15));

            stockPriceLabel.Text = String.Format("{0:#,###}", stockPrice);
            stockUpDownLabel.Text = String.Format("{0:#,###}", upDown);
            stockVolumeLabel.Text = String.Format("{0:#,###}", volume);
            if (upDown == 0)
            {
                stockUpDownLabel.Text = "0";
            }
            if (volume == 0)
            {
                stockVolumeLabel.Text = "0";
            }
            stockUpDownRateLabel.Text = upDownRate + "%";
        }
    }
}

```

4.1에서 작성했던 OnReceiveTrData 함수와 유사한 형태입니다.

```
axKHOpenAPI1.GetCommRealData(종목코드, Fid번호);
```

아래 코드는 전체 소스코드입니다.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp3
{
    public partial class Form1 : Form
    {
        List<stockInfo> stockList;

        public string currentStockCode = "";
        public Form1()
        {
            InitializeComponent();
            axKHOpenAPI1.CommConnect();
            axKHOpenAPI1.OnEventConnect += onEventConnect;
            axKHOpenAPI1.OnReceiveRealData += onReceiveRealData;
            axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
            stockSearchButton.Click += stockSearch;
        }

        public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
        {
            if (e.sRQName=="종목정보요청")
            {
                l          o          n          g
                stockPrice=long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,e.sRQName,0,"현재가"
").Trim().Replace("-", ""));

                string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "종목명").Trim();

                long upDown = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "전일대비").Trim());
                long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "거래량").Trim());
                string upDownRate = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "등락율").Trim();
            }
        }
}
```

```

stockPriceLabel.Text = String.Format("{0:#,###}", stockPrice);
stockNameLabel.Text = stockName;
stockUpDownLabel.Text = String.Format("{0:#,###}", upDown);
stockVolumeLabel.Text = String.Format("{0:#,###}", volume);
if (upDown == 0)
{
    stockUpDownLabel.Text = "0";
}
if (volume == 0)
{
    stockVolumeLabel.Text = "0";
}
stockUpDownRateLabel.Text = upDownRate + "%";
}

public void stockSearch(object sender, EventArgs e)
{
    string stockName = stockTextBox.Text;
    int index = stockList.FindIndex(o => o.stockName == stockName);
    string stockCode = stockList[index].stockCode;
    currentStockCode = stockCode;

    axKHOpenAPI1.SetInputValue("종목코드", stockCode);
    axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
}

public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    if (e.sRealType == "주식체결")
    {
        if (e.sRealKey == currentStockCode)
        {
            long stockPrice = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode, 10));
            long upDown = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode, 11));
            string upDownRate = axKHOpenAPI1.GetCommRealData(currentStockCode, 12);
            long volume = long.Parse(axKHOpenAPI1.GetCommRealData(currentStockCode, 15));
        }
    }
}

```

```

        stockPriceLabel.Text = String.Format("{0:#,###}", stockPrice);
        stockUpDownLabel.Text = String.Format("{0:#,###}", upDown);
        stockVolumeLabel.Text = String.Format("{0:#,###}", volume);
        if (upDown == 0)
        {
            stockUpDownLabel.Text = "0";
        }
        if (volume == 0)
        {
            stockVolumeLabel.Text = "0";
        }
        stockUpDownRateLabel.Text = upDownRate + "%";
    }
}
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
    stockList = new List<stockInfo>();

    if (e.nErrCode==0)
    {
        AutoCompleteStringCollection stockcollection = new
AutoCompleteStringCollection();

        string stockCode= axKHOpenAPI1.GetCodeListByMarket(null);
        string [] stockCodeArray = stockCode.Split('\'');
        for (int i = 0; i < stockCodeArray.Length; i++)
        {
            stockList.Add(new stockInfo(stockCodeArray[i],
axKHOpenAPI1.GetMasterCodeName(stockCodeArray[i])));
        }
        for (int i = 0; i < stockList.Count; i++)
        {
            stockcollection.Add(stockList[i].stockName);
        }
        stockTextBox.AutoCompleteCustomSource = stockcollection;
    }
}
}

```

```
class stockInfo
{
    public string stockCode;
    public string stockName;
    public stockInfo(string code,string name)
    {
        this.stockCode = code;
        this.stockName = name;
    }
}
```

실행화면입니다.

현대로템1

Form1 window titled "현대로템". The "종목검색" button is highlighted with a blue border. A table displays the following stock information:

종목명	현대로템
현재가	31,850
전일대비	4,850
거래량	142
등락률	+17.96%

An "OpenAPI" button is located at the bottom of the form.

현대로템2

Form1 window titled "현대로템". The "종목검색" button is highlighted with a blue border. A table displays the following stock information:

종목명	현대로템
현재가	31,800
전일대비	4,800
거래량	1
등락률	+17.78%

An "OpenAPI" button is located at the bottom of the form.

프로그램을 실행하면 실시간으로 종목 정보가 변경되는 것을 확인 할 수 있습니다.
5장에서는 호가창을 만드는 방법에 대해서 알아보도록 하겠습니다.

5. 호가창

Preview

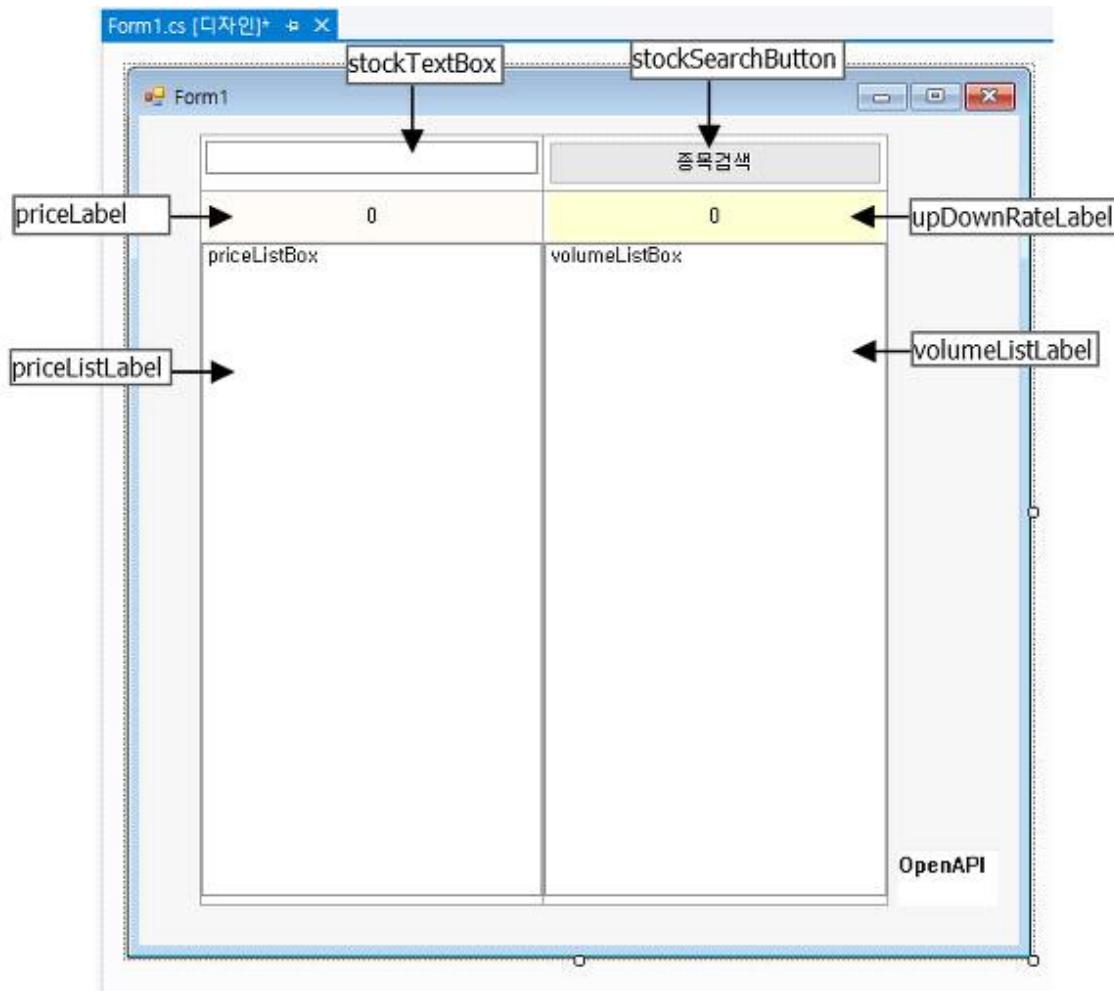
5장에서는 호가창을 만드는 방법에 대해서 알아보도록 하겠습니다. 호가창은 실시간으로 투자자들의 호가 정보를 알아볼 수 있는 화면입니다.

Contents

5.1장에서는 hts의 호가창을 만드는 방법을 알아봅니다. 호가창을 만들면서 화면에 색을 칠하는 방법도 함께 알아봅니다. DrawItem과 DrawMode를 사용해봅니다. 화면에 특정 목록을 출력하는 ListBox에 대해서도 학습합니다.

5.1 호가창

5장에서는 호가창을 만드는 방법에 대해서 알아보도록 하겠습니다. 호가창은 실시간으로 투자자들의 가격별 주문량을 알아볼 수 있는 화면입니다. 프로젝트를 하나 생성하고 화면을 구성합니다.



stockTextBox의 속성

속성	값
(Name)	stockTextBox
Anchor	Top, Bottom, Left, Right
AutoCompleteMode	Suggest
AutoCompleteSource	CustomSource

stockSearchButton의 속성

속성	값
(Name)	stockSearchButton
Anchor	Top, Bottom, Left, Right
Text	종목검색
TextAlign	MiddleCenter

priceLabel의 속성

속성	값
(Name)	priceLabel
Anchor	Top, Bottom, Left, Right
BackColor	SeaShell
Text	0
.TextAlign	MiddleCenter

upDownRateLabel의 속성

속성	값
(Name)	upDownRateLabel
Anchor	Top, Bottom, Left, Right
BackColor	LemonChiffon
Text	0
.TextAlign	MiddleCenter

priceListBox의 속성

속성	값
(Name)	priceListBox
Anchor	Top, Bottom, Left, Right
Margin	0, 0, 0, 0

volumeListBox의 속성

속성	값
(Name)	volumeListBox
Anchor	Top, Bottom, Left, Right
Margin	0, 0, 0, 0

stockTextBox 는 종목의 이름을 입력할 TextBox입니다. 4.1, 4.2장에서 만들었던 것처럼 만들면 됩니다.

사용자가 로그인 했을때 호출되는 이벤트 함수인 OnEventConnect 함수에서 stockTextBox 의 자동완성 기능을 구현합니다. 호가창을 만들기 위해서 종목을 검색했을때 호가정보와 가격 정보를 요청하도록 하겠습니다. 종목검색 버튼을 클릭했을때 호가정보, 가격정보를 요청하도록 코드를 작성하겠습니다. CommRqData함수를 사용해서 정보 요청을 보냅니다. 가격 정보를 먼저 요청해보도록 하겠습니다. 4.1장에서 코드를 작성했던 것처럼 아래와 같이 요청을 보내면 됩니다.

그리고 요청을 보내기 전에 stockTextBox안에 있는 종목 이름을 통해서 알아낸 종목 코드가 비어있지 않은지 확인해야합니다.

```
public void stockSearch(object sender, EventArgs e)
{
    string stockName = stockTextBox.Text;
    int index = stockList.FindIndex(o => o.stockName == stockName);
    currentStockCode = stockList[index].stockCode;

    if (currentStockCode.Length > 0)
    {
        axKHOpenAPI1.SetInputValue("종목코드", currentStockCode);
        axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
    }
}
```

```
axKHOpenAPI1.SetInputValue("종목코드", currentStockCode);
axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
```

종목 정보를 요청했다면 호가정보도 요청해보도록 하겠습니다.

```
axKHOpenAPI1.SetInputValue("종목코드", currentStockCode);
axKHOpenAPI1.CommRqData("주식호가", "opt10004", 0, "5001");
```

요청을 보낼때 종목정보 요청과 주식 호가 요청의 화면번호는 달라야 합니다.

이렇게 종목검색 버튼을 클릭했을때 두가지 요청이 전송되도록 코드를 작성했다면 요청의 결과로 호출될 이벤트 함수를 작성합니다.

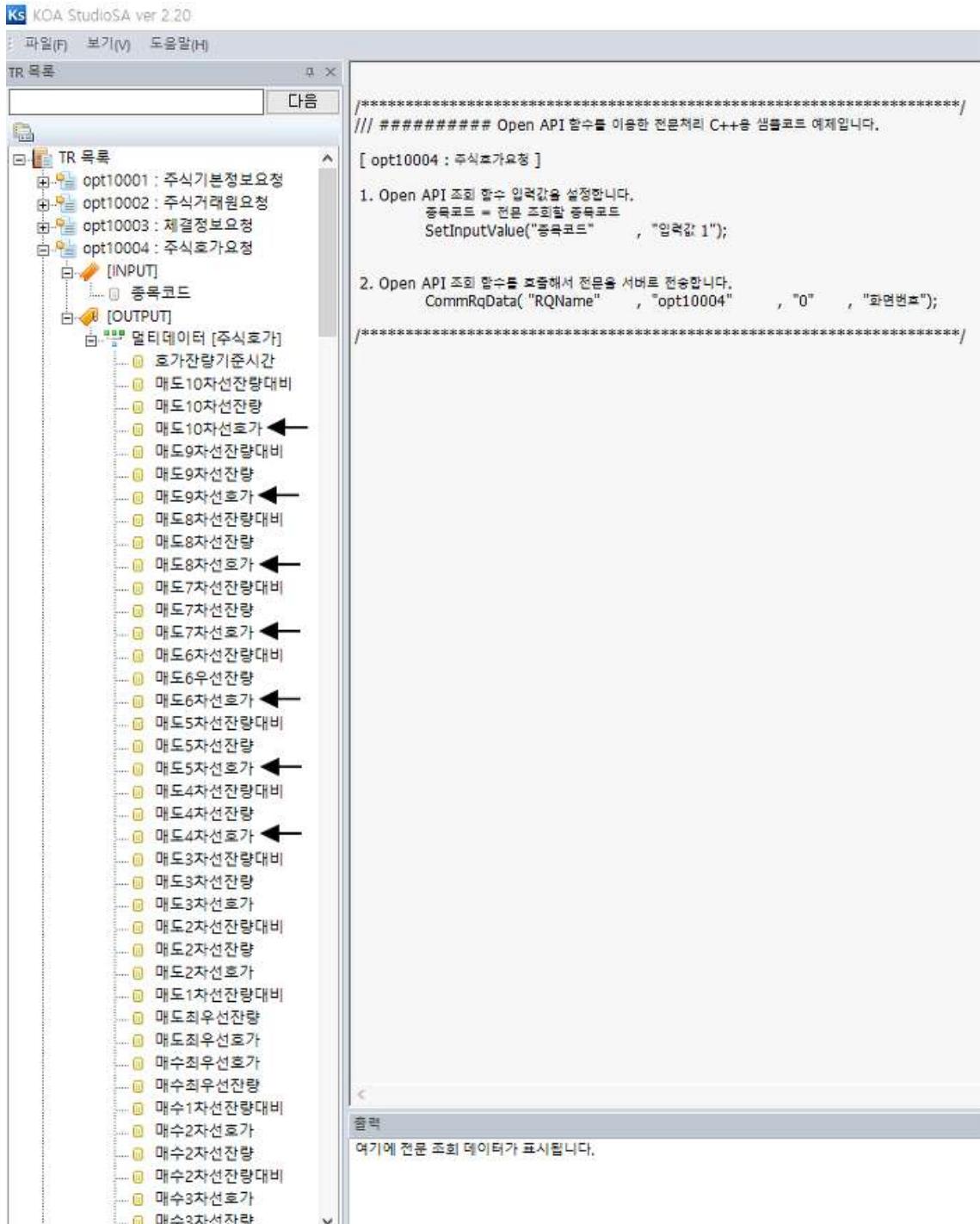
```
axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
axKHOpenAPI1.OnReceiveRealData += onReceiveRealData;
```

OnReceiveTrData 함수의 내용부터 작성하도록 하겠습니다. 이전에 사용자는 CommRqData 함수를 호출해서 두가지 요청을 보냈습니다. "주식호가" 요청을 OnReceiveTrData 함수에서 if 문으로 나눠서 처리하도록 하겠습니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPIlib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
    if (e.sRQName == "주식호가")
    {
    }
}
```

priceLabel과 upDownRateLabel은 OnReceiveRealData 함수에서 실시간 데이터를 받은 다음에 바인딩하도록 하겠습니다.

이제 if문 안의 내용을 작성하면 됩니다. GetCommData를 사용해서 매수, 매도호가 정보를 요청합니다.



OnReceiveTrData 함수 안에서 GetCommData 함수를 KOA Studio를 참고해서 작성합니다. 아래 코드처럼 GetCommData함수의 인자값으로 TrCode , RQName , 0 , Output이름을 작성하면 됩니다.

```
axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도2차선호가");
```

e.sTrCode 에는 사용자가 CommRqData로 요청한 TR 코드의 번호가 들어있습니다. e.sRQName 에는 사용자가 CommRqData로 요청했던 사용자 요청 이름이 들어있습니다. 0 은 연속 조회 여부를 나타냅니다. "매도2차선호가"는 주식호가요청 Tr 데이터의 output이름입니다. 데이터를 요청할때는 input을 작성해서 이벤트 함수를 호출하고 이벤트 함수 안에서 output을 작성해서 데이터를 요청합니다.

매수,매도 호가의 output에는 규칙성이 있습니다.

매수2차선, 매수3차선 , 매수4차선 , 매수5차선 매도2차선, 매도3차선 , 매도4차선 , 매도5차선

데이터를 요청 할 때 이런 규칙성을 활용해보도록 하겠습니다.

for문을 사용해서 규칙성을 이용하면 됩니다.

```
for (int i = 0; i < 9; i++)
{
    int 차선호가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "매도" + (10 - i) + "차선호가"));
    priceListBox.Items.Add(차선호가 + " / " + (10 - i) + "차선");

    if (i == 4)
    {
        int 차선잔량 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "매도" + (10 - i) + "우선잔량"));
        volumeListBox.Items.Add(차선잔량);
    }
    else
    {
        int 차선잔량 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "매도" + (10 - i) + "차선잔량"));
        volumeListBox.Items.Add(차선잔량);
    }
}
```

호가 요청의 결과를 listBox에 바인딩 합니다. listBox에 items.add를 사용해서 데이터를 추가하면 한줄씩 데이터가 추가됩니다.

opt10001 TR 요청은 "주식체결" 실시간 타입을 주로 불러옵니다. opt10004 TR 요청은 "주식호가잔량" 실시간 타입을 주로 불러옵니다.

OnReceiveRealData함수에서 "주식체결"과 "주식호가잔량"에 대한 처리를 if문으로 구분하여 작성합니다.

```
public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
    if (e.sRealType == "주식호가잔량")
    {

    }
    else if (e.sRealType == "주식체결")
    {

    }
}
```

if(주식호가잔량)에서는 주식 호가와 거래량이 실시간으로 변경되는 것을 화면에 바인딩 합니다. else if(주식체결)에서는 실시간으로 주가, 등락율이 변경되는 것을 화면에 바인딩 합니다. if(주식호가잔량)에서는 GetCommRealData함수를 실시간타입(Real Type:주식호가잔량)을 참고해서 작성합니다. else if(주식체결)에서는 GetCommRealData함수를 실시간타입(Real Type:주식체결)을 참고해서 작성합니다. 지금 하고 있는 것은 GetCommData로 요청해서 바인딩한 초기 데이터를 GetCommRealData로 요청한 데이터로 변경하는 것입니다.

```
if (e.sRealKey.Equals(currentStockCode))
{
    try
    {
        for (int i = 0; i < 10; i++)
        {
            int 매도호가 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType,
50 - i));
            int 매도잔량 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType,
70 - i));

            priceListBox.Items[i] = 매도호가 + " / +"(10-i)+"차선";
            volumeListBox.Items[i] = 매도잔량;

            if (i==9)
            {
                priceListBox.Items[i] = 매도호가 + " / 매도최우선호가";
            }
        }
    }
}
```

```
        }
    }
    for (int i = 0; i < 10; i++)
    {
        int 매수호가 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType,
51 + i));
        int 매수잔량 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType,
71 + i));

        priceListBox.Items[10 + i] = 매수호가 + " / " + (i+1) + "차선";
        volumeListBox.Items[10 + i] = 매수잔량;

        if (i==0)
        {
            priceListBox.Items[10 + i] = 매수호가 + " / 매수최우선호가";
        }
    }
    catch (Exception exception)
    {
        Console.WriteLine(exception.Message.ToString());
    }
}
```

if(주식호가잔량) 아래에 이 코드를 작성합니다. if(e.sRealkey)에는 OnReceiveRealData 이벤트 함수가 호출 되기 이전에 보냈던 종목 코드가 담겨있습니다. 종목검색을 통해서 검색했던 종목 코드와 e.sRealKey에 담겨있는 종목코드가 동일하다면 호가요청을 실시간으로 보냅니다. else if(주식체결) e.sRealKey에 담긴 종목코드와 검색했던 종목코드가 같다면 현재가, 등락율을 실시간으로 요청합니다.

```
    axKHOOpenAPI1.GetCommRealData(e.sRealType, 10);
```

주식체결도 GetCommRealData함수를 사용해서 요청을 보냅니다.

여기까지 코드를 작성하면 화면에서 호가 정보와 현재가,등락율이 변동되는 것을 확인 할 수 있습니다. 실제 호가창과 더 비슷하게 만들기 위해서 DrawItem 함수를 사용하도록 하겠습니다.

```
priceListBox.DrawItem += ListBox_DrawItem;
priceListBox.DrawMode = DrawMode.OwnerDrawVariable;

priceListBox.ItemHeight = priceListBox.Height / 20;

volumeListBox.DrawItem += ListBox_DrawItem;
volumeListBox.DrawMode = DrawMode.OwnerDrawVariable;
volumeListBox.ItemHeight = volumeListBox.Height / 20;
```

Form1.cs의 생성자에 코드를 추가합니다. 함수이름을 추가하고 함수의 내용을 작성합니다.

```
public void ListBox_DrawItem(object sender, DrawItemEventArgs e)
{
}
```

DrawItem 함수의 내용은 호가창에 색칠을 하는 것입니다. priceListBox 와 volumeListBox 를 나눠서 코드를 작성하도록 하겠습니다.

```
if (sender.Equals(priceListBox))
{
}
else if (sender.Equals(volumeListBox))
{
}
```

이벤트 객체 e의 타입 DrawItemEventArgs 클래스에는 Index , Graphics 와 같은 여러 필드들이 있습니다. Index를 사용해서 listbox의 행에 접근 할 수 있습니다. Graphics를 사용해서 listbox 에 색칠을 하거나 그림을 그릴 수 있습니다. 그래서 Index를 사용해서 화면의 20개의 호가 가격중에서 10번째 가격까지는 배경을 파랗게 칠하고, 11~20번째 가격의 배경은 붉게 칠하도록 하겠습니다.

```
e.Graphics.FillRectangle(Brushes.LightSteelBlue,      new      Rectangle(e.Bounds.X,
e.Bounds.Y, e.Bounds.Width, e.Bounds.Height));
```

FillRectangle 함수를 호출하면 사각형을 그릴 수 있습니다. 코드를 이렇게 작성하면 listbox 의 한 행마다 푸른 색 사각형을 그리게 됩니다.

붉은 색 사각형은 아래와 같이 작성합니다.

```
e.Graphics.FillRectangle(Brushes.LightPink,      new      Rectangle(e.Bounds.X,
e.Bounds.Y, e.Bounds.Width, e.Bounds.Height));
```

다음은 listBox의 값을 받아오고 값이 -로 시작하면 글씨를 파랗게 칠하도록 하겠습니다. 주식 가격에 -가 붙어있지 않다면 빨갛게 칠하도록 하겠습니다.

```
String value = priceListBox.Items[e.Index].ToString();
Brush brush;

if (value[0] == '-')
{
    brush = Brushes.Blue;
}
else
{
    brush = Brushes.Red;
}
```

이렇게 리스트 박스의 인덱스를 입력해서 숫자를 가져오고, value[0] 으로 맨 앞의 문자가 -인지 확인합니다. -이면 brush를 blue로 저장합니다.

글씨를 쓰기위한 DrawString 함수를 호출하기 위해서 코드를 더 작성합니다. DrawString 함수는 원래 인자가 아래와 같이 정의되어 있습니다.

```
public void DrawString(string s, Font font, Brush brush, float x, float y,
StringFormat format);
```

s는 적을 문자열입니다. 적을 문자열을 맨 앞에 적습니다. font는 글꼴입니다. 여기서는 별도로 지정하지 않았습니다. brush는 그려지는 텍스트의 색과 질감을 결정합니다. 여기서는 Brush.Blue, Brush.Red로 지정했습니다. x와 y는 그려지는 텍스트의 왼쪽 위 모퉁이의 x,y 좌표입니다. format은 그려지는 텍스트에 적용될 줄 간격, 맞춤 같은 서식 특성을 지정하는 부분입니다. 여기서는 StringFormat.GenericDefault로 지정했습니다. 글씨를 그릴 때 제일 기본형태로 지정했습니다 GenericDefault말고 아래 코드처럼 Typographic도 있습니다. 두 옵션 사이에 큰 차이는 없습니다.

```
StringFormat.GenericTypographic
```

양식에 맞춰서 아래 코드처럼 인자를 작성하고 함수를 호출합니다.

```
int x = e.Bounds.X + e.Font.Height / 2;
int y = e.Bounds.Y + e.Font.Height / 2;

e.Graphics.DrawString(value.Replace("-", ""), e.Font, brush, x, y,
StringFormat.GenericDefault);
```

e.Bounds.X는 왼쪽 위 모퉁이의 X좌표입니다. 기본값은 0입니다. e.Font.Height는 글꼴의 줄간격입니다. DrawString의 인자로 보낼 x를 e.Bounds.X와 e.Font.Height를 사용해서 지정하고 있습니다. x를 e.Bounds.X로만 지정해서 0으로 한다면 글씨가 왼쪽에 바짝 붙어서 출력되게 됩니다. 그래서 줄 간격의 절반을 더해줬습니다. y는 왼쪽 위 모퉁이의 y좌표입니다. x처럼 기본값은 0입니다. y좌표가 0이라면 글씨가 왼쪽 위에 바짝 붙어서 출력되게 됩니다. y에 Height를 2로 나눈 값을 지정합니다. 이렇게 왼쪽 위 모퉁이로부터 적당히 글씨가 떨어지도록 할 수 있습니다.

아래코드는 호가창 프로그램의 전체 코드입니다.

전체코드

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApp4
{
    public partial class Form1 : Form
    {
        List<stockInfo> stockList;
        public string currentStockCode = "";
        public Form1()
        {
            InitializeComponent();
            axKHOpenAPI1.CommConnect();
            axKHOpenAPI1.OnEventConnect += onEventConnect;
            axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
            axKHOpenAPI1.OnReceiveRealData += onReceiveRealData;
            stockSearchButton.Click += stockSearch;

            priceListBox.DrawItem += ListBox_DrawItem;
            priceListBox.DrawMode = DrawMode.OwnerDrawVariable;

            priceListBox.ItemHeight = priceListBox.Height / 20;

            volumeListBox.DrawItem += ListBox_DrawItem;
            volumeListBox.DrawMode = DrawMode.OwnerDrawVariable;
            volumeListBox.ItemHeight = volumeListBox.Height / 20;
        }

        public void ListBox_DrawItem(object sender, DrawItemEventArgs e)
        {
            if (sender.Equals(priceListBox))
            {
                try
                {
                    if (e.Index < 10)
                    {
                        e.Graphics.FillRectangle(Brushes.LightSteelBlue, new
```
```

```

 }
 else
 {
 e.Graphics.FillRectangle(Brushes.LightPink, new
Rectangle(e.Bounds.X, e.Bounds.Y, e.Bounds.Width, e.Bounds.Height));
 }
 String value = priceListBox.Items[e.Index].ToString();
 Brush brush;

 if (value[0] == '-')
 {
 brush = Brushes.Blue;
 }
 else
 {
 brush = Brushes.Red;
 }
 int x = e.Bounds.X + e.Font.Height / 2;
 int y = e.Bounds.Y + e.Font.Height / 2;

 e.Graphics.DrawString(value.Replace("-", ""), e.Font, brush, x,
y, StringFormat.GenericDefault);
 }
 catch (Exception exception)
 {
 Console.WriteLine(exception.Message.ToString());
 }
}
else if (sender.Equals(volumeListBox))
{
 try
 {
 string value = volumeListBox.Items[e.Index].ToString();

 int x = e.Bounds.X + e.Font.Height / 2 + e.Bounds.Width /
2;
 int y = e.Bounds.Y + e.Font.Height / 2;

 e.Graphics.DrawString(value, e.Font, Brushes.Black, x, y,
StringFormat.GenericDefault);
 }
}

```

```

 catch (Exception exception)
 {
 Console.WriteLine(exception.Message.ToString());
 }
 }

 public void stockSearch(object sender , EventArgs e)
 {
 string stockName = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockName);
 currentStockCode = stockList[index].stockCode;

 if (currentStockCode.Length>0)
 {
 axKHOpenAPI1.SetInputValue("종목코드", currentStockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");

 axKHOpenAPI1.SetInputValue("종목코드", currentStockCode);
 axKHOpenAPI1.CommRqData("주식호가", "opt10004", 0, "5001");
 }
 }

 public void onReceiveTrData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "주식호가")
 {
 priceListBox.Items.Clear();
 volumeListBox.Items.Clear();
 for (int i = 0; i < 9; i++)
 {
 int 차선호가 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도" + (10 - i)
+ "차선호가"));
 priceListBox.Items.Add(차선호가 + " / " + (10 - i) + "차선");

 if (i == 4)
 {
 }
 }
}

```

```

 int 차선잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도" + (10 - i)
+ "우선잔량"));
 volumeListBox.Items.Add(차선잔량);
}

else
{
 int 차선잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도" + (10 - i)
+ "차선잔량"));
 volumeListBox.Items.Add(차선잔량);
}

}

int 매도최우선호가 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도최우선호가"
));
 priceListBox.Items.Add(매도최우선호가 + " / 매도최우선호가");
 int 매도최우선잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매도최우선잔량"
));
 volumeListBox.Items.Add(매도최우선잔량);

 int 매수최우선호가 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수최우선호가"
));
 priceListBox.Items.Add(매수최우선호가 + " / 매수최우선호가");
 int 매수최우선잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수최우선잔량"
));
 volumeListBox.Items.Add(매수최우선잔량);

 for (int i = 0; i < 9; i++)
 {
 if (i == 4)
 {
 int 호가 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수" + (2 + i) +
"우선호가"));
 priceListBox.Items.Add(호가 + " / " + (2 + i) + "차선");
 }
 }
}

```

```

 int 잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수" + (2 + i) +
"우선잔량"));
 volumeListBox.Items.Add(잔량);
}
else
{
 int 호가 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수" + (2 + i) +
"차선호가"));
 priceListBox.Items.Add(호가 + " / " + (2 + i) + "차선");
 int 잔량 =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수" + (2 + i) +
"차선잔량"));
 volumeListBox.Items.Add(잔량);
}
}

MessageBox.Show("매도최우선호가 = " + 매도최우선호가 + "매수최우
선호가 = " + 매수최우선호가);
}

}

public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
 Console.WriteLine(currentStockCode
"-----+");
 if (e.sRealType == "주식호가잔량")
 {
 if (e.sRealKey.Equals(currentStockCode))
 {
 try
 {
 for (int i = 0; i < 10; i++)
 {
 int 매도호가 =
int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 50 - i));
 int 매도잔량 =
int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 70 - i));
 }
 }
 }
 }
}

```

```

 priceListBox.Items[i] = 매도호가 + " / "+(10-i)+"차선";
 volumeListBox.Items[i] = 매도잔량;

 if (i==9)
 {
 priceListBox.Items[i] = 매도호가 + " / 매도최우선호
 가";
 }

 }

 for (int i = 0; i < 10; i++)
 {
 int 매수호가 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 51 + i));
 int 매수잔량 = int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 71 + i));

 priceListBox.Items[10 + i] = 매수호가 + " / " + (i+1) +
 "차선";
 volumeListBox.Items[10 + i] = 매수잔량;

 if (i==0)
 {
 priceListBox.Items[10 + i] = 매수호가 + " / 매수최
 우선호가";
 }

 }

}

catch (Exception exception)
{
 Console.WriteLine(exception.Message.ToString());
}

}

}

else if (e.sRealType == "주식체결")
{
 if (e.sRealKey.Equals(currentStockCode))
}

```

```

 {
 long stockPrice = long.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 10).Replace("-", ""));
 priceLabel.Text = String.Format("{0:#,###}", stockPrice);

 upDownRateLabel.Text = double.Parse(axKHOpenAPI1.GetCommRealData(e.sRealType, 12)) + "%";
 }
}

public void onEventConnect(object sender, AxKHOpenAPIlib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 stockList = new List<stockInfo>();
 if (e.nErrCode == 0)
 {
 string stockCodeList = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockCodeArray = stockCodeList.Split(';');
 AutoCompleteStringCollection stockcollection = new AutoCompleteStringCollection();
 for (int i = 0; i < stockCodeArray.Length; i++)
 {
 s t o c k L i s t . A d d (n e w stockInfo(stockCodeArray[i],axKHOpenAPI1.GetMasterCodeName(stockCodeArray[i])));
 stockcollection.Add(stockList[i].stockName);
 }
 stockTextBox.AutoCompleteCustomSource = stockcollection;
 }
}

class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockcode,string stockname)
 {
 this.stockCode = stockcode;
 this.stockName = stockname;
 }
}

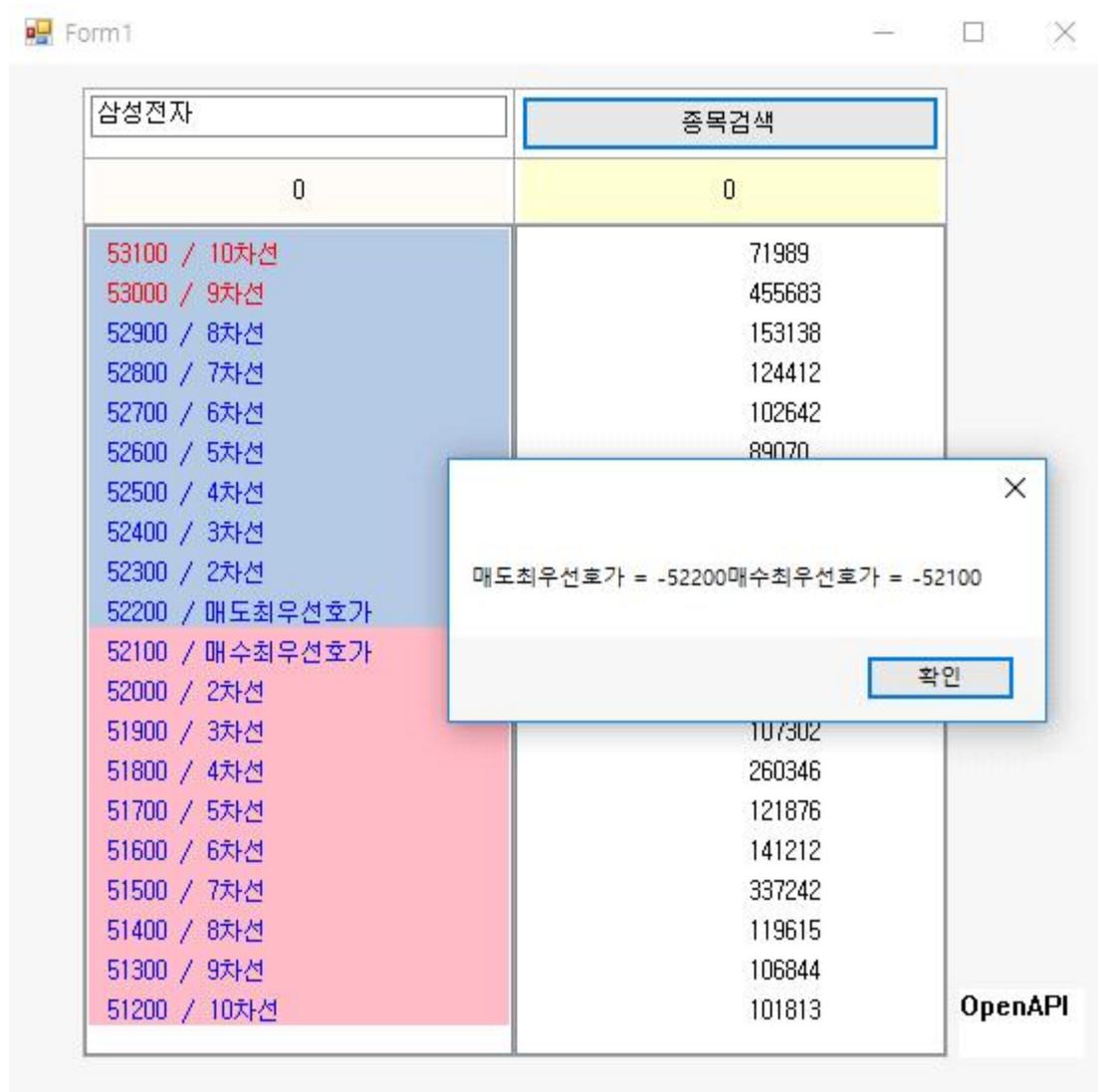
```

```

 }
 }
}

```

## 실행화면



| 삼성전자            | 종목검색   |
|-----------------|--------|
| 52,100          | -1.7%  |
| 53000 / 10차선    | 464869 |
| 52900 / 9차선     | 150499 |
| 52800 / 8차선     | 121553 |
| 52700 / 7차선     | 100396 |
| 52600 / 6차선     | 95184  |
| 52500 / 5차선     | 89996  |
| 52400 / 4차선     | 85805  |
| 52300 / 3차선     | 56316  |
| 52200 / 2차선     | 97909  |
| 52100 / 매도최우선호가 | 72758  |
| 52000 / 매수최우선호가 | 436842 |
| 51900 / 2차선     | 122156 |
| 51800 / 3차선     | 266337 |
| 51700 / 4차선     | 123277 |
| 51600 / 5차선     | 146671 |
| 51500 / 6차선     | 336168 |
| 51400 / 7차선     | 120069 |
| 51300 / 8차선     | 106764 |
| 51200 / 9차선     | 99890  |
| 51100 / 10차선    | 125689 |

OpenAPI

Form1

| 삼성전자            | 증목검색   |
|-----------------|--------|
| 52,000          | -1.89% |
| 53000 / 10차선    | 465169 |
| 52900 / 9차선     | 150499 |
| 52800 / 8차선     | 121553 |
| 52700 / 7차선     | 100406 |
| 52600 / 6차선     | 95091  |
| 52500 / 5차선     | 89051  |
| 52400 / 4차선     | 85754  |
| 52300 / 3차선     | 54743  |
| 52200 / 2차선     | 97832  |
| 52100 / 매도최우선호가 | 83498  |
| 52000 / 매수최우선호가 | 434278 |
| 51900 / 2차선     | 122404 |
| 51800 / 3차선     | 266671 |
| 51700 / 4차선     | 123116 |
| 51600 / 5차선     | 146951 |
| 51500 / 6차선     | 335906 |
| 51400 / 7차선     | 120059 |
| 51300 / 8차선     | 106832 |
| 51200 / 9차선     | 99907  |
| 51100 / 10차선    | 125693 |

OpenAPI

## 6. 매수 주문

### Preview

6장에서는 프로그램에서 매수주문을 요청합니다. 매수 주문을 요청하는 방법은 자동매매 프로그램을 만들 때 중요합니다. 매수 금액에 제한을 설정하는 방법도 알아봅니다.

### Contents

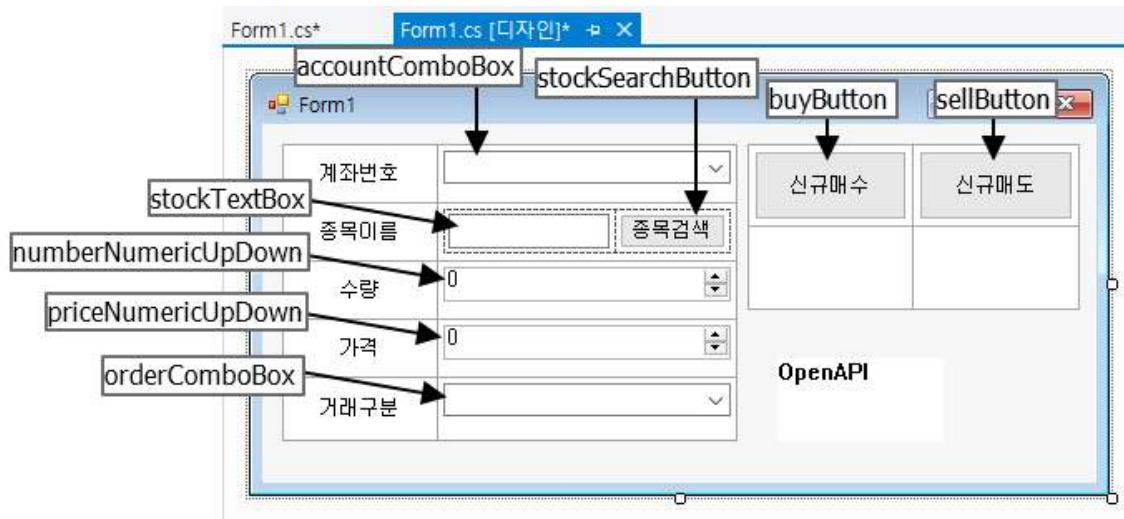
6.1장에서는 프로그램에서 주식을 매수합니다. 매수주문을 요청하는 SendOrder 함수에 대해서 학습합니다. SendOrder함수로 매수 주문을 요청할 때 필요한 정보들도 알아봅니다.

6.2장에서는 프로그램에서 매수 제한 금액을 설정해봅니다. 사용자가 설정한 금액을 초과하여 매수 주문이 되지 않도록 금액 제한을 설정합니다. 3.3장에서 학습한 전체 매입금액 정보를 활용합니다.

## 6.1 프로그램에서 매수 주문하기

6.1장에서는 프로그램에서 주식 매수주문을 하는 방법에 대해서 알아보도록 하겠습니다.

아래 그림처럼 화면을 구성합니다.



### accountComboBox의 속성

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | accountComboBox          |
| Anchor | Top, Bottom, Left, Right |

### stockTextBox의 속성

|                    |                          |
|--------------------|--------------------------|
| 속성                 | 값                        |
| (Name)             | stockTextBox             |
| AutoCompleteMode   | Suggest                  |
| AutoCompleteSource | CustomSource             |
| Anchor             | Top, Bottom, Left, Right |

### stockSearchButton의 속성

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | stockSearchButton        |
| Text      | 종목검색                     |
| TextAlign | MiddleCenter             |
| Anchor    | Top, Bottom, Left, Right |

### numberNumericUpDown의 속성

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | numberNumericUpDown      |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

### priceNumericUpDown의 속성

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | priceNumericUpDown       |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**orderComboBox의 속성**

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | orderComboBox            |
| Anchor | Top, Bottom, Left, Right |

**buyButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | buyButton                |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매수                     |
| TextAlign | MiddleCenter             |

**sellButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | sellButton               |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매도                     |
| TextAlign | MiddleCenter             |

화면을 모두 구성했다면 코드를 작성합니다. 6.1장에서는 아래와 같은 순서로 코드를 작성하게 됩니다.

로그인 -> 화면 구성요소 바인딩 -> 신규매수 Button 기능 구현  
Form1.cs에 로그인 함수와 OnEventConnect 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
}
```

OnEventConnect 함수의 동작을 정의합니다. OnEventConnect 함수에서는 화면 구성요소를 바인딩 하도록 하겠습니다. 매수주문을 하는 함수는 SendOrder 함수입니다.

|                                                         |
|---------------------------------------------------------|
| [SendOrder() 함수]                                        |
| SendOrder(                                              |
| BSTR sRQName, // 사용자 구분명                                |
| BSTR sScreenNo, // 화면번호                                 |
| BSTR sAccNo, // 계좌번호 10자리                               |
| LONG nOrderType, // 주문유형 1:신규매수, 2:신규매도 3:매수취소, 4:매도취소, |
| 5:매수정정, 6:매도정정                                          |
| BSTR sCode, // 종목코드                                     |
| LONG nQty, // 주문수량                                      |
| LONG nPrice, // 주문가격                                    |
| BSTR sHogaGb, // 거래구분(혹은 호가구분)은 아래 참고                   |
| BSTR sOrgOrderNo // 원주문번호입니다. 신규주문에는 공백, 정정(취소)주문할      |
| 원주문번호를 입력합니다.                                           |
| )                                                       |

9개 인자값을 가진 국내 주식주문 함수이며 리턴값이 0이면 성공이며 나머지는 에러입니다.

1초에 5회만 주문가능하며 그 이상 주문요청하면 에러 -308을 리턴합니다.

#### [거래구분]

모의투자에서는 지정가 주문과 시장가 주문만 가능합니다.

00 : 지정가

03 : 시장가

05 : 조건부지정가

매수주문을 하려면 계좌번호, 종목코드, 거래구분 정보가 화면에 바인딩 되어 있어야 합니다. 주문수량, 주문가격은 사용자가 화면에서 작성 할 수 있습니다. 사용자구분명, 화면 번호, 주문유형은 프로그램에서 소스코드로 작성 할 수 있습니다.

OnEventConnect 함수에 관련내용을 정의해보도록 하겠습니다.

```
public void onEventConnect(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());
 }
}
```

"00:지정가", "03:시장가" 문자열을 화면에 바인딩하는 과정입니다. SendOrder 함수에서 거래구분에 해당하는 부분입니다. 거래구분의 값은 00, 03 등의 문자열로 입력해줘야 합니다. 화면에 "00:지정가", "03:시장가"처럼 문자열을 바인딩 해놓으면 종목을 주문할때 코드에서 :(콜론)을 기준으로 문자열을 분리하고 사용 할 수 있습니다.

orderComboBox의 내용을 바인딩 했다면 이번에는 계좌번호를 바인딩해야 합니다. GetLoginInfo 함수를 호출해서 정보를 요청하고 계좌정보를 바인딩 합니다.

```
string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
string[] accountArray = accountList.Split(';');
for (int i = 0; i < accountArray.Length; i++)
{
 accountComboBox.Items.Add(accountArray[i]);
}
```

계좌정보를 모두 바인딩 했다면, 종목 자동완성 목록을 만듭니다. 4.1장의 종목 검색에서 했던 것과 같은 내용입니다.

```
public partial class Form1 : Form
{
 List<stockInfo> stockList;
}

Form1 클래스에 리스트를 만들고, 종목 코드를 요청합니다. 요청의 결과로 얻은 종목코드를 사용해서 종목의 이름을 알아냅니다. 장내의 모든 종목코드, 종목이름을 AutoCompleteStringCollection에 추가합니다.

string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(':');
AutoCompleteStringCollection stockCollection = new AutoCompleteStringCollection();

for (int i = 0; i < stockArray.Length; i++)
{
 stockList.Add(new stockInfo(stockArray[i],
 axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
}
stockTextBox.AutoCompleteCustomSource = stockCollection;
```

계좌번호, 자동완성목록, 거래구분을 모두 화면에 바인딩 했다면 OnEventConnect 함수의 내용을 모두 작성한 것입니다. 종목검색 Button을 클릭했을때 종목의 현재 가격정보가 화면에 바인딩 되도록 코드를 작성하도록 하겠습니다. 종목의 현재 가격정보를 화면에 바인딩 하면 사용자가 주문 가격을 설정하기 편리합니다. 종목검색 Button을 클릭했을때 동작할 함수를 Form1.cs의 생성자에 추가합니다.

```
stockSearchButton.Click += stockSearch;
stockSearch 함수의 내용을 작성합니다. stockSearch함수에서는 stockTextBox에 적힌 종목이름을 가지고 종목의 현재가를 알아내야 합니다.
```

```
public void stockSearch(object sender, EventArgs e)
{
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
}
```

시장에 있는 종목들의 목록을 OnEventConnect 함수에서 stockList에 추가했습니다. stockSearch 함수에서는 stockText에 적힌 종목이름과 stockList에 있는 종목 이름을 대조하여 종목 코드를 찾아내면 됩니다. 알아낸 종목코드를 사용해서 종목정보를 요청합니다. CommRqData 함수를 호출해서 정보를 요청하면 됩니다. CommRqData함수를 호출해서 요청한 정보는 OnReceiveTrData 함수에서 확인 할 수 있습니다.

Form1.cs의 생성자에 OnReceiveTrData 함수를 추가하고 함수의 내용을 작성합니다.

```
axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
```

onReceiveTrData 함수 안에서 GetCommData 함수를 호출해서 종목의 현재 가격 정보를 요청합니다. 종목의 현재 가격 정보에는 '-' 문자가 포함되어 있을 수도 있습니다. 전일의 가격 대비 가격이 하락했다면 현재 가격에 -문자가 포함됩니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
}
```

화면의 priceNumericUpDown에 현재 가격 정보를 바인딩 합니다.

가격정보를 바인딩 했다면 신규매수 Button 을 클릭했을때 매수주문을 하는 코드를 작성하도록 하겠습니다.

```
buyButton.Click += orderButtonClicked;
```

Form1.cs의 생성자에 함수를 추가하고 함수의 내용을 작성합니다.

```
public void orderButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 && accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode, 1,
orderCode, stockQty, stockPrice, orderCombo[0], "");
 }
 }
}
```

```

 }
}

```

먼저 if문을 사용해서 stockTextBox 가 비어있는지 확인합니다. stockTextBox가 비어있다면 종목코드를 찾을 수 없습니다. accountComboBox 의 값도 들어있는지 체크합니다. 사용자가 작성한 numberNumericUpDown , priceNumericUpDown의 값도 모두 가져옵니다. SendOrder함수에 인자 값을 차례대로 적고 매수주문을 보냅니다.

## 전체코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp8
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 stockSearchButton.Click += stockSearch;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 buyButton.Click += orderButtonClicked;
 }

 public void orderButtonClicked(object sender, EventArgs e)
 {
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 &&
 accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty =

```

```

int.Parse(numberNumericUpDown.Value.ToString());
int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
string[] orderCombo = orderComboBox.Text.Split(':');

axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode,
1, orderCode, stockQty, stockPrice, orderCombo[0], "");

}

}

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
}

public void stockSearch(object sender, EventArgs e)
{
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
}

public void onEventConnect(object sender, ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 }
}

```

```
{
 accountComboBox.Items.Add(accountArray[i]);
}
string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(';');
AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

for (int i = 0; i < stockArray.Length; i++)
{
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));

stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
}
stockTextBox.AutoCompleteCustomSource = stockCollection;
}
}
}
}
class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode, string stockName)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}
}
```

## 실행화면

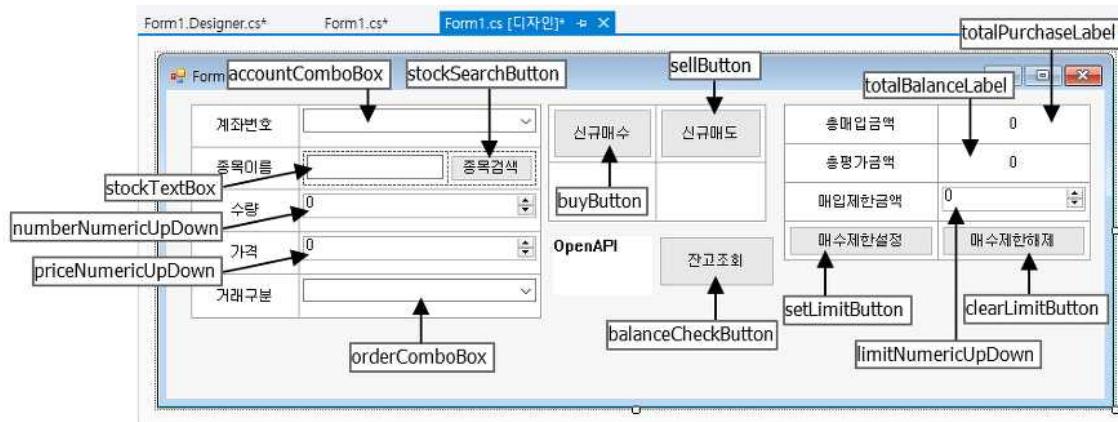
## HTS에서 확인한 화면

| 총매입 | 6,701,835 | 총손익     | 1,466  | 실현손익    | 2,179 | 일괄매도 |         |
|-----|-----------|---------|--------|---------|-------|------|---------|
| 총평가 | 6,770,650 | 총수익률    | 0.02%  | 추정자산    |       | 조회   |         |
|     | 종목명       | 평가손익    | 수익률    | 매입가     | 보유수량  | 가능수량 | 현재가     |
|     | 태양금속우     | -484    | -1.07% | 4,515   | 10    | 10   | 4,510   |
|     | 삼성전자      | -16,916 | -1.10% | 51,450  | 30    | 30   | 51,400  |
|     | 오뚜기       | 18,646  | 0.42%  | 732,500 | 6     | 6    | 743,000 |
|     | 네이처셀      | -3,535  | -1.16% | 30,550  | 10    | 10   | 30,500  |
|     | 한국내화      | -1,088  | -2.32% | 4,688   | 10    | 10   | 4,625   |
|     | 까뮤이앤씨     | 936     | 5.51%  | 1,700   | 10    | 10   | 1,810   |

조회가 완료 되었습니다.

## 6.2 매수금액 제한 걸기

6.2 장에서는 6.1장의 매수주문에 금액 제한을 설정하는 방법을 다룹니다. 6.1장의 매수주문하기를 참고해서 화면을 구성합니다.



6.2장의 목적은 사용자가 프로그램에 제한 금액을 설정하고 해당 금액 이상 매수주문을 진행 할 수 없도록 하는 것입니다.

오른쪽 TableLayout 과 잔고조회 Button을 추가했습니다. 화면 구성 컨트롤들의 속성은 아래와 같습니다.

### accountComboBox의 속성

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | accountComboBox          |
| Anchor | Top, Bottom, Left, Right |

### stockTextBox의 속성

|                    |                          |
|--------------------|--------------------------|
| 속성                 | 값                        |
| (Name)             | stockTextBox             |
| AutoCompleteMode   | Suggest                  |
| AutoCompleteSource | CustomSource             |
| Anchor             | Top, Bottom, Left, Right |

### stockSearchButton의 속성

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | stockSearchButton        |
| Text      | 종목검색                     |
| TextAlign | MiddleCenter             |
| Anchor    | Top, Bottom, Left, Right |

### numberNumericUpDown의 속성

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | numberNumericUpDown      |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**priceNumericUpDown의 속성**

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | priceNumericUpDown       |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**orderComboBox의 속성**

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | orderComboBox            |
| Anchor | Top, Bottom, Left, Right |

**buyButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | buyButton                |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매수                     |
| TextAlign | MiddleCenter             |

**sellButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | sellButton               |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매도                     |
| TextAlign | MiddleCenter             |

**balanceCheckButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | balanceCheckButton       |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 잔고조회                     |
| TextAlign | MiddleCenter             |

**totalBalanceLabel의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | totalBalanceLabel        |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 0                        |
| TextAlign | MiddleCenter             |

**totalPurchaseLabel의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | totalPurchaseLabel       |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 0                        |
| TextAlign | MiddleCenter             |

**limitNumericUpDown의 속성**

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | limitNumericUpDown       |
| Anchor  | Top, Bottom, Left, Right |
| Maximum | 10000000                 |

**setLimitUpDown의 속성**

|            |                          |
|------------|--------------------------|
| 속성         | 값                        |
| (Name)     | setLimitUpDown           |
| Anchor     | Top, Bottom, Left, Right |
| Text       | 매수제한설정                   |
| .TextAlign | MiddleCenter             |

**clearLimitUpDown의 속성**

|            |                          |
|------------|--------------------------|
| 속성         | 값                        |
| (Name)     | clearLimitUpDown         |
| Anchor     | Top, Bottom, Left, Right |
| Text       | 매수제한해제                   |
| .TextAlign | MiddleCenter             |

이제 코드를 작성해보도록 하겠습니다. 6.2장의 코드 작성 순서는 아래와 같습니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 총 매입금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

화면에 계좌번호 , 거래구분 , 자동완성 목록을 바인딩 하도록 하겠습니다. 사용자 로그인 이후에 거래구분 , 계좌번호 , 자동완성 목록이 화면에 바인딩 되면 됩니다. Form1.cs 의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
}
```

로그인 함수와 OnEventConnect 함수를 추가했습니다. OnEventConnect 함수에서 화면에 초기 정보를 바인딩하면 됩니다.

```
public void onEventConnect(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());
 }
}
```

거래구분 정보를 먼저 바인딩합니다. 거래구분 정보는 사용자가 정보를 요청하지 않아도 쉽게 바인딩 할 수 있습니다.

이제 아래와 같이 코드를 작성해서 계좌번호 정보를 요청해서 바인딩 합니다 4.1장 , 6.1장에서도 작성했었던 내용입니다.

```
string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
string[] accountArray = accountList.Split(':');
for (int i = 0; i < accountArray.Length; i++)
{
 accountComboBox.Items.Add(accountArray[i]);
}
```

종목 자동완성 목록도 만들어서 바인딩 합니다. 종목 자동완성 목록을 만들고 사용하기 위해서는 stockTextBox의 AutoCompleteMode , AutoCompleteSource 속성을 변경해줘야 합니다. Form1.cs[디자인] 화면에서 AutoComplete 속성이 잘 변경되었는지 확인 한 후 코드를 작성합니다.

```
string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(':');
AutoCompleteStringCollection stockCollection = new AutoCompleteStringCollection();

for (int i = 0; i < stockArray.Length; i++)
{
 stockList.Add(new stockInfo(stockArray[i],
 axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
}
stockTextBox.AutoCompleteCustomSource = stockCollection;
```

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 총 매입금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

신규매수 Button을 클릭했을 때 동작할 코드를 작성합니다. 매수 제한을 위해서 Form1.cs의 클래스에 변수를 두개 선언하고 코드 작성은 진행합니다.

```
public partial class Form1 : Form
{
 List<stockInfo> stockList;
 long buyingLimit;
 long purchasedAmount;
```

buyingLimit 은 매수 제한 설정 button을 클릭했을때 매수제한금액이 담길 변수입니다. purchasedAmount는 사용자의 당일 매수 금액을 저장할 변수입니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 buyButton.Click += ButtonClicked;
}
```

Form1.cs 의 생성자에 함수를 추가합니다. buyButton.Click 에 함수를 추가하면 됩니다. orderButtonClicked 함수의 내용을 작성합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
}
```

if문을 사용해서 buyButton에서 일어난 Event 인지 확인합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 }
}
```

ordereButtonClicked 함수의 매개변수 sender에는 발생한 Event를 요청한 개체의 정보가 들어있습니다. sender 가 buyButton 과 일치하면 함수의 다음 동작을 진행합니다. 매수주문을 요청하기 위해서는 계좌번호 , 종목코드가 필수입니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 && accountComboBox.Text.Length > 0)
 {
 }
 }
}
```

if 문의 조건으로 종목코드를 찾기 위한 종목 이름이 비어있는지 확인합니다. accountComboBox 도 값이 들어있는지 확인해서 선택한 계좌번호 정보를 가져옵니다. 값이 잘 들어있는지 if문으로 체크했다면 값을 가져오는 코드를 작성합니다.

| 값    | 출처                  |
|------|---------------------|
| 종목명  | stockTextBox        |
| 계좌번호 | accountComboBox     |
| 수량   | numberNumericUpDown |
| 가격   | priceNumericUpDown  |

아래와 같이 화면의 구성요소를 호출하는 코드를 작성합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 && accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode, 1,
orderCode, stockQty, stockPrice, orderCombo[0], "");
 }
 }
}
```

화면의 값을 가져왔다면 SendOrder함수를 호출해서 매수주문을 합니다. 6.1장에서 작성했던 코드와 같습니다. 여기서는 금액 제한을 위해서 매수주문 코드를 조금 변경합니다.

```
if (sender.Equals(buyButton))
{
 if (stockTextBox.Text.Length > 0 && accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');
```

```

if ((buyingLimit==0) ||(buyingLimit > purchasedAmount))
{
 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode, 1,
orderCode, stockQty, stockPrice, orderCombo[0], "");
}
axKHOpenAPI1.SetInputValue("계좌번호", accountCode);
axKHOpenAPI1.SetInputValue("비밀번호", "2351");
axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
axKHOpenAPI1.CommRqData("계좌별당일현황요청", "opw00017", 0, "7100");
}
}

```

매수를 할때마다 계좌별 당일 현황을 요청합니다. 그리고 buyingLimit의 값이 purchasedAmount의 값보다 클 경우에만 SendOrder 함수가 호출 되도록 코드를 변경했습니다. buyingLimit의 값이 0일때에도 SendOrder 함수가 호출되도록 코드를 작성했습니다. buyingLimit의 값이 0인 경우에 대해서는 6.2장의 마지막 부분에서 다루겠습니다. 매수금액 제한 해제에 관한 내용입니다. 이렇게 코드를 작성하면 매수 버튼을 클릭할때마다 새롭게 당일 매수 금액을 요청 할 수 있습니다. CommRqData함수는 이후 작성할 OnReceiveTrData 함수를 호출하게 됩니다. OnReceiveTrData 함수에서는 매수금액을 요청하고 Form1.cs의 변수에 저장할 것입니다. 여기까지 코드를 작성하면 신규매수 Button 의 코드가 모두 작성 된 것입니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 총 매입금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

잔고조회 Button에 대한 코드를 작성할 차례입니다. 7.1장에서는 잔고조회 Button을 클릭하면 사용자가 가지고 있는 주식 잔고가 나타나도록 코드를 작성할 계획입니다. 6.2장에서는 잔고조회 Button 을 클릭하면 사용자의 당일매수금액 , 총평가금액을 화면에 바인딩 하도록 하겠습니다.

ButtonClicked 함수를 작성할때 sender 에 담긴 요청 개체를 if 문으로 구분 할 수 있다고 했습니다.

```

public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

 }
 else if (sender.Equals(balanceCheckButton))
 {

```

```

 }
}

```

잔고조회 Button은 else if 문을 작성해서 처리합니다. 이벤트가 balanceCheckButton에서 일어났을 때 처리할 코드를 else if 문 안에 작성합니다. 잔고 조회 요청은 3.3장의 계좌평 가현황 요청을 참고해서 코드를 작성하면 됩니다.

```

public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

 }
 else if (sender.Equals(balanceCheckButton))
 {
 string accountNumber = accountComboBox.Text;
 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.SetInputValue("조회구분", "1");
 axKHOpenAPI1.CommRqData("계좌평가잔고내역요청", "opw00018", 0, "8100");
 }
}

```

당일 매수 금액도 함께 요청합니다.

```

.....
.....
.....
else if (sender.Equals(balanceCheckButton))
{
 string accountNumber = accountComboBox.Text;
 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.SetInputValue("조회구분", "1");
 axKHOpenAPI1.CommRqData("계좌평가잔고내역요청", "opw00018", 0, "8100");

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌별당일현황요청", "opw00017", 0, "7100");
}

```

SetInputValue 함수에 필요한 인자값들을 모두 작성하고 CommRqData 함수를 호출합니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 총 매입금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

CommRqData 함수를 호출해서 보낸 요청은 OnReceiveTrData 함수를 사용해서 확인 할 수 있으므로 OnReceiveTrData 함수를 작성합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 buyButton.Click += ButtonClicked;
 balanceCheckButton.Click += ButtonClicked;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
}
```

OnReceiveTrData 함수를 Form1.cs의 생성자에 추가하고 함수의 내용을 작성합니다.

```
public void onReceiveTrData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e){}
```

if문과 e.sRQName을 사용해서 요청을 구분하고 코드를 작성합니다.

```
public void onReceiveTrData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e){
if (e.sRQName == "계좌평가잔고내역요청")
{
 long totalBalance = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총평가금액"));
 totalBalanceLabel.Text = String.Format("{0:#,###}", totalBalance);
}}
```

GetCommData 함수를 호출하고 함수의 인자 값으로 TR목록의 Output 값을 적습니다. 이렇게 코드를 작성하면 totalBalanceLabel에 총평가금액을 요청해서 바인딩 할 수 있습니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 총 매입금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

이제 매수제한설정 Button을 클릭했을때 동작할 코드를 작성하겠습니다. Form1.cs에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 buyButton.Click += ButtonClicked;
 balanceCheckButton.Click += ButtonClicked;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 setLimitButton.Click += ButtonClicked;
}
```

setLimitButton.Click에도 ButtonClicked 함수를 추가합니다. ButtonClicked 함수 안에서 else if로 구분해서 함수의 내용을 정의합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

 }
 else if (sender.Equals(balanceCheckButton))
 {

 }
 else if (sender.Equals(setLimitButton))
 {
 ...
 }
}
```

매수 제한 금액을 설정하기 위해서 Form1.cs 의 클래스에 변수를 하나 선언합니다. setLimitButton을 클릭하면 NumericUpDown 컨트롤의 값을 가져와서 매수제한 금액을 설정하면 됩니다.

```
public partial class Form1 : Form
{
 ...
 long buyingLimit;
 ...
}
```

선언한 변수는 setLimit Button이 클릭되었을때 초기화 하도록 합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

 }
 else if (sender.Equals(balanceCheckButton))
 {

 }
 else if (sender.Equals(setLimitButton))
 {
 if (long.Parse(limitNumericUpDown.Value.ToString()) > 0)
 {
 ...
 }
 }
}
```

아래와 같이 코드를 작성해서 buyingLimit 변수를 초기화합니다.

```
.....
.....
else if (sender.Equals(setLimitButton))
{
 if (long.Parse(limitNumericUpDown.Value.ToString()) > 0)
 {
 buyingLimit = long.Parse(limitNumericUpDown.Value.ToString());
 Console.WriteLine(buyingLimit);
 MessageBox.Show("금액제한 설정완료" + buyingLimit + "원");
 }
}
```

변수를 초기화 하는 코드를 작성하기 전에 limitNumericUpDown.Value가 0보다 큰지 확인합니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 매수금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

이제 매수제한해제 Button함수를 작성하도록 하겠습니다. 매수금액 제한 해제는 buyingLimit의 값을 0으로 초기화 하는 것으로 할 수 있습니다. 신규매수 Button의 동작을 정의할때 buyingLimit의 값이 0이어도 SendOrder 함수가 호출되도록 코드를 작성했기 때문입니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 buyButton.Click += ButtonClicked;
 balanceCheckButton.Click += ButtonClicked;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 setLimitButton.Click += ButtonClicked;
 clearLimitButton.Click += ButtonClicked;
}
```

ButtonClicked 함수에 else if문으로 작성하면 됩니다.

```
else if (sender.Equals(clearLimitButton))
{
 buyingLimit = 0;
 MessageBox.Show("매수금액제한 해제");
}
```

buyingLimit 값이 0이면 SendOrder함수가 정상 동작 할 수 있도록 이전의 if 문을 작성했습니다. 이렇게 코드를 작성하면 buyingLimit 값을 0으로 초기화 하는 것으로 매수금액 제한 을 해제 할 수 있습니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 신규매수 Button 코드 작성
- 잔고조회 Button 코드 작성
- 매수금액 , 총 평가금액 바인딩
- 매수제한설정 Button 함수 작성
- 매수제한해제 Button 함수 작성

## 전체 코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp11
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 long buyingLimit;
 long purchasedAmount;
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 buyButton.Click += ButtonClicked;
 balanceCheckButton.Click += ButtonClicked;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 setLimitButton.Click += ButtonClicked;
 clearLimitButton.Click += ButtonClicked;
 }

 public void onReceiveTrData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e){
 if (e.sRQName == "계좌평가잔고내역요청")
 {
 long totalBalance =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "총평가금액"));
 totalBalanceLabel.Text = String.Format("{0:#,###}", totalBalance);
 }
 else if (e.sRQName=="계좌별당일현황요청")
 {
 purchasedAmount =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "매수금액"));
 todayPurchaseLabel.Text = String.Format("{0:#,###}",
purchasedAmount);
 }
 }

 public void ButtonClicked(object sender, EventArgs e)
 }
}

```

```

 {
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 &&
accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 if ((buyingLimit==0) ||(buyingLimit > purchasedAmount))
 {
 axKHOpenAPI1.SendOrder("종목신규매수", "8249",
accountCode, 1, orderCode, stockQty, stockPrice, orderCombo[0], "");
 }
 axKHOpenAPI1.SetInputValue("계좌번호", accountCode);
 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌별당일현황요청", "opw00017",
0, "7100");
 }
 }
 else if (sender.Equals(balanceCheckButton))
 {
 string accountNumber = accountComboBox.Text;
 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.SetInputValue("조회구분", "1");
 axKHOpenAPI1.CommRqData("계좌평가잔고내역요청", "opw00018", 0,
"8100");

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 }
 }
}

```

```

 axKHOpenAPI1.SetInputValue("비밀번호", "2351");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌별당일현황요청", "opw00017", 0,
 "7100");
 }

 else if (sender.Equals(setLimitButton))
 {
 if (long.Parse(limitNumericUpDown.Value.ToString()) > 0)
 {
 buyingLimit =
long.Parse(limitNumericUpDown.Value.ToString());
 Console.WriteLine(buyingLimit);
 MessageBox.Show("금액제한 설정완료" + buyingLimit + "원");
 }
 }

 else if (sender.Equals(clearLimitButton))
 {
 buyingLimit = 0;
 MessageBox.Show("매수금액제한 해제");
 }
}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection =
new

```

```
AutoCompleteStringCollection():

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));

 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 }

 stockTextBox.AutoCompleteCustomSource = stockCollection;

}

}

}

class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode , string stockName)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}
```

## 7. 매도 주문하기

### Preview

7장에서는 프로그램에서 매도 주문을 요청합니다. 매도 주문을 요청하기 위해서는 사용자가 가지고 있는 주식 잔고를 확인해야 합니다. 사용자의 주식 잔고를 확인하는 방법도 소개합니다.

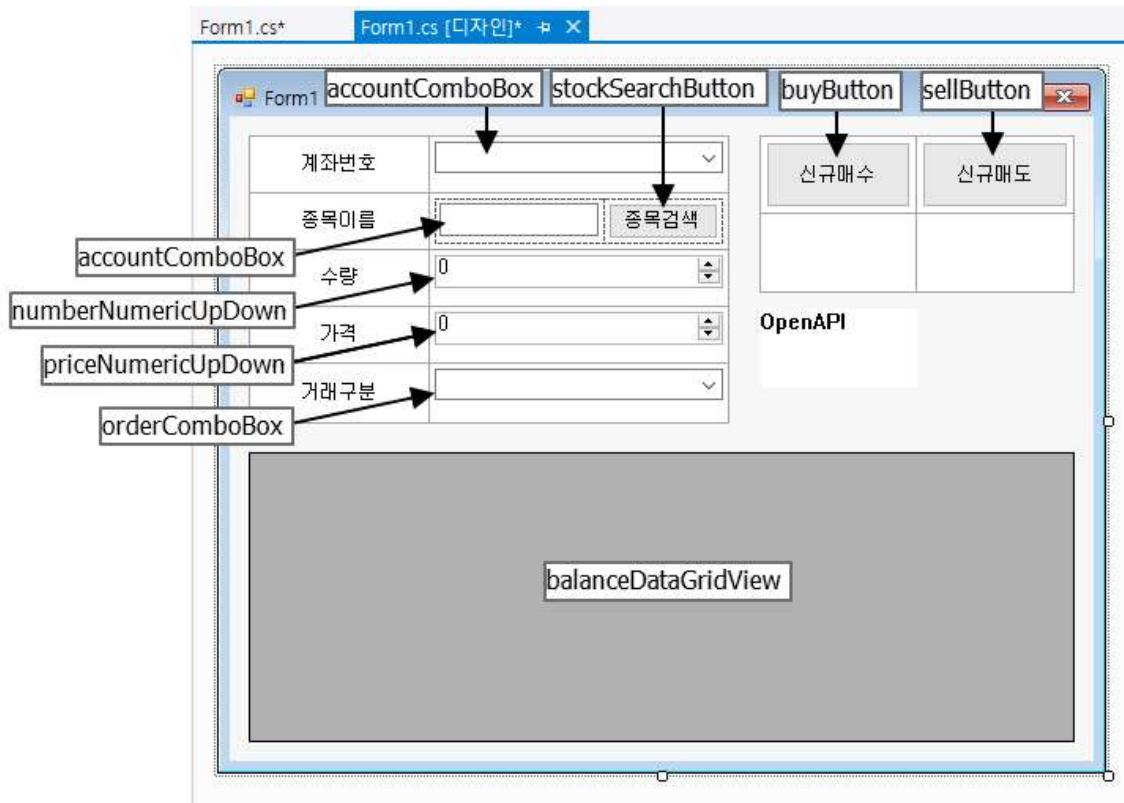
### Contents

7.1장에서는 사용자의 주식 잔고를 확인합니다. OnReceiveTrData 함수를 활용합니다.

7.2장에서는 주식 매도 주문을 요청하는 방법을 알아봅니다. SendOrder 함수를 사용해서 매도 주문을 요청합니다.

## 7.1 사용자 주식 잔고 확인

7.1장에서는 사용자가 보유하고 있는 주식잔고를 확인하도록 하겠습니다. 먼저 화면을 구성합니다. 6.1장에서 구성한 화면과 매우 유사합니다.



사용자 계좌 잔고를 표시할 `balanceDataGridView`가 추가되었습니다.

화면의 구성요소는 아래와 같습니다.

### accountComboBox의 속성

| 속성     | 값                        |
|--------|--------------------------|
| (Name) | accountComboBox          |
| Anchor | Top, Bottom, Left, Right |

### stockTextBox의 속성

| 속성                 | 값                        |
|--------------------|--------------------------|
| (Name)             | stockTextBox             |
| AutoCompleteMode   | Suggest                  |
| AutoCompleteSource | CustomSource             |
| Anchor             | Top, Bottom, Left, Right |

### stockSearchButton의 속성

| 속성        | 값                        |
|-----------|--------------------------|
| (Name)    | stockSearchButton        |
| Text      | 종목검색                     |
| TextAlign | MiddleCenter             |
| Anchor    | Top, Bottom, Left, Right |

**numberNumericUpDown의 속성**

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | numberNumericUpDown      |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**priceNumericUpDown의 속성**

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | priceNumericUpDown       |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**orderComboBox의 속성**

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | orderComboBox            |
| Anchor | Top, Bottom, Left, Right |

**buyButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | buyButton                |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매수                     |
| TextAlign | MiddleCenter             |

**sellButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | sellButton               |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매도                     |
| TextAlign | MiddleCenter             |

**balanceDataGridView의 속성**

|                  |                     |
|------------------|---------------------|
| 속성               | 값                   |
| (Name)           | balanceDataGridView |
| RowHeaderVisible | False               |
| SelectionMode    | FullRowSelect       |

7.1장에서는 다음과 같은 순서로 코드를 작성합니다.

로그인 -> 화면구성요소 바인딩 -> 잔고조회 함수 작성

Form1.cs 의 생성자에 함수를 추가하고 순서대로 코드를 작성하도록 하겠습니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect(); //로그인 함수
 axKHOpenAPI1.OnEventConnect += onEventConnect; //로그인이 되었을때 호출되는 이벤트함수
 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer; //조건식 요청
 conditionDataGridView.SelectionChanged += conditionSearch; //조건식 검색
 axKHOpenAPI1.OnReceiveTrCondition += onReceiveTrCondition; //조건식 검색결과
}
```

함수들을 모두 추가했다면 하나씩 함수의 내용을 작성해보겠습니다. OnEventConnect 함수의 내용부터 작성하면 됩니다. 항상 OnEventConnect 함수에는 프로그램을 실행 했을때 바인딩 될 내용을 호출하는 코드를 작성하면 좋습니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 잔고조회 Button 코드 작성

프로그램을 실행 했을때 화면에 바인딩되어야 하는 것은 3가지입니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가").ToString());
 orderComboBox.Items.Add("03:시장가").ToString());
 }
}
```

거래 구분 문자열을 화면에 바인딩 합니다. 7.1장은 잔고조회 Button의 동작을 정의해서 주식 잔고를 확인 하는 것이 목적이므로 이 장에서는 중요한 부분은 아닙니다.

```
stockList = new List<stockInfo>();
string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
string[] accountArray = accountList.Split(';');
for (int i = 0; i < accountArray.Length; i++)
{
 accountComboBox.Items.Add(accountArray[i]);
}
string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(';');
```

```
AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();
```

계좌번호와 자동완성 목록도 완성해서 화면에 추가합니다. stockList는 Form1.cs 의 클래스 변수로 선언합니다.

아래 코드는 OnEventConnect 함수의 전체 코드입니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(':');
 AutoCompleteStringCollection stockCollection = new
 AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
 }
}
```

- 화면에 계좌번호, 거래구분, 자동완성 목록 바인딩
- 잔고조회 Button 코드 작성

이제 잔고조회 Button의 코드를 작성합니다. Form1.cs의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 buyButton.Click += orderButtonClicked;
 stockSearchButton.Click += stockSearch;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
}
```

balanceCheckButton을 클릭하면 동작할 balanceCheck 함수를 Form1.cs의 생성자에 추가합니다. 함수의 내용을 작성합니다.

```
public void balanceCheck(object sender, EventArgs e)
{
 accountNumber = accountComboBox.Text;
 string password = "";//계좌비밀번호 입력

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "0004");
}
```

balanceCheckButton을 클릭하면 CommRqData 함수를 호출해서 계좌평가현황을 요청합니다. 이제 OnReceiveTrData 함수에서 요청을 확인하고 보유한 종목정보를 요청하면 됩니다. 아래 KOA Studio의 예시처럼 코드를 작성하면 됩니다.

```
[GetCommData() 함수]
GetCommData(
 BSTR strTrCode, // TR 이름
 BSTR strRecordName, // 레코드이름
 long nIndex, // TR반복부
 BSTR strItemName) // TR에서 얻어오려는 출력항목이름
```

OnReceiveTRData()이벤트 함수가 호출될때 조회데이터를 얻어오는 함수입니다.

이 함수는 반드시 OnReceiveTRData()이벤트 함수가 호출될때 그 안에서 사용해야 합니다.

[OPT10081 : 주식일봉차트조회요청예시]

```

OnReceiveTrData()
{
 if(strRQName == T("주식일봉차트"))
 {
 int nCnt = OpenAPI.GetRepeatCnt(sTrcode, strRQName);
 for (int nIdx = 0; nIdx < nCnt; nIdx++)
 {
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("종목코드"));
 strData.Trim();
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("거래량"));
 strData.Trim();
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("시가"));
 strData.Trim();
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("고가"));
 strData.Trim();
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("저가"));
 strData.Trim();
 strData = OpenAPI.GetCommData(sTrcode, strRQName, nIdx, T("현재가"));
 strData.Trim();
 }
 }
}

```

CommRqData 함수에 사용자 구분명을 "계좌평가현황요청" 이라고 적었습니다.  
OnReceiveTrData 함수 안에서 else if 로 요청을 구분하고 코드를 작성합니다.

```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
 else if (e.sRQName == "계좌평가현황요청")

```

```

{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "보유수량"));
 long buyingMoney =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 long currentPrice =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));
 long estimatedProfit =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));
 double estimatedProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalanceList.Add(new stockBalance(stockCode, stockName,
number, String.Format("{0:#,###}", buyingMoney), String.Format("{0:#,###}",
currentPrice), estimatedProfit, String.Format("{0:f2}", estimatedProfitRate)));
 }
 balanceDataGridView.DataSource = stockBalanceList;
}
}

```

요청의 결과를 받았다면 변수에 저장한 뒤에 balanceDataGridView에 바인딩 합니다.

## 전체 코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp9
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 List<stockBalance> stockBalanceList;
 public string accountNumber = "";
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 buyButton.Click += orderButtonClicked;
 sellButton.Click += orderButtonClicked;
 stockSearchButton.Click += stockSearch;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
 balanceDataGridView.SelectionChanged +=
 bindingByBalanceDataGridView;
 }

 public void bindingByBalanceDataGridView(object sender, EventArgs e)
 {
 if (balanceDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 string[] stockPriceArray = balanceDataGridView["현재가",
 rowIndex].Value.ToString().Split(',');
 string stockPrice = "";
 for (int i = 0; i < stockPriceArray.Length; i++)
 {
 stockPrice = stockPrice + stockPriceArray[i];
 }
 long stockCurrentPrice = long.Parse(stockPrice);
 priceNumericUpDown.Value = stockCurrentPrice;
 }
 }
 }
}

```

```

 }

 }

 public void balanceCheck(object sender, EventArgs e)
 {
 accountNumber = accountComboBox.Text;
 string password = "";//사용자 비밀번호 입력

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "0004");
 }

 public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
 else if (e.sRQName == "계좌평가현황요청")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "보유수량"));
 long buyingMoney = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));
 long estimatedProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));
 }
 }
}

```

```

long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));
 double estimatedProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalanceList.Add(new stockBalance(stockCode,
stockName, number, String.Format("{0:#,###}", buyingMoney),
String.Format("{0:#,###}", currentPrice), estimatedProfit, String.Format("{0:f2}",
estimatedProfitRate)));
}
balanceDataGridView.DataSource = stockBalanceList;
}
}

public void stockSearch(object sender, EventArgs e)
{
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
}

public void orderButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 &&
accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty =
int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice =
int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode,
1, orderCode, stockQty, stockPrice, orderCombo[0], "");
 }
 }
}

```

```

 }

 }

}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
 }
}

class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode, string stockName)
 {

```

```
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}

class stockBalance
{
 public string 종목코드 { get; set; }
 public string 종목명 { get; set; }
 public long 수량 { get; set; }
 public string 매수금 { get; set; }
 public string 현재가 { get; set; }
 public long 평가손익 { get; set; }
 public string 수익률 { get; set; }

 public stockBalance() { }

 public stockBalance(string 종목번호, string 종목명, long 수량, string 매수
금, string 현재가, long 평가손익, string 수익률)
 {
 this.종목코드 = 종목번호;
 this.종목명 = 종목명;
 this.수량 = 수량;
 this.매수금 = 매수금;
 this.현재가 = 현재가;
 this.평가손익 = 평가손익;
 this.수익률 = 수익률;
 }
}
```

## 사용자주식잔고확인

## 실행화면

Form1

|      |                           |
|------|---------------------------|
| 계좌번호 | 8101317911                |
| 종목이름 | <input type="text"/> 종목검색 |
| 수량   | 0                         |
| 가격   | 4445                      |
| 거래구분 | <input type="text"/>      |

신규매수

신규매도

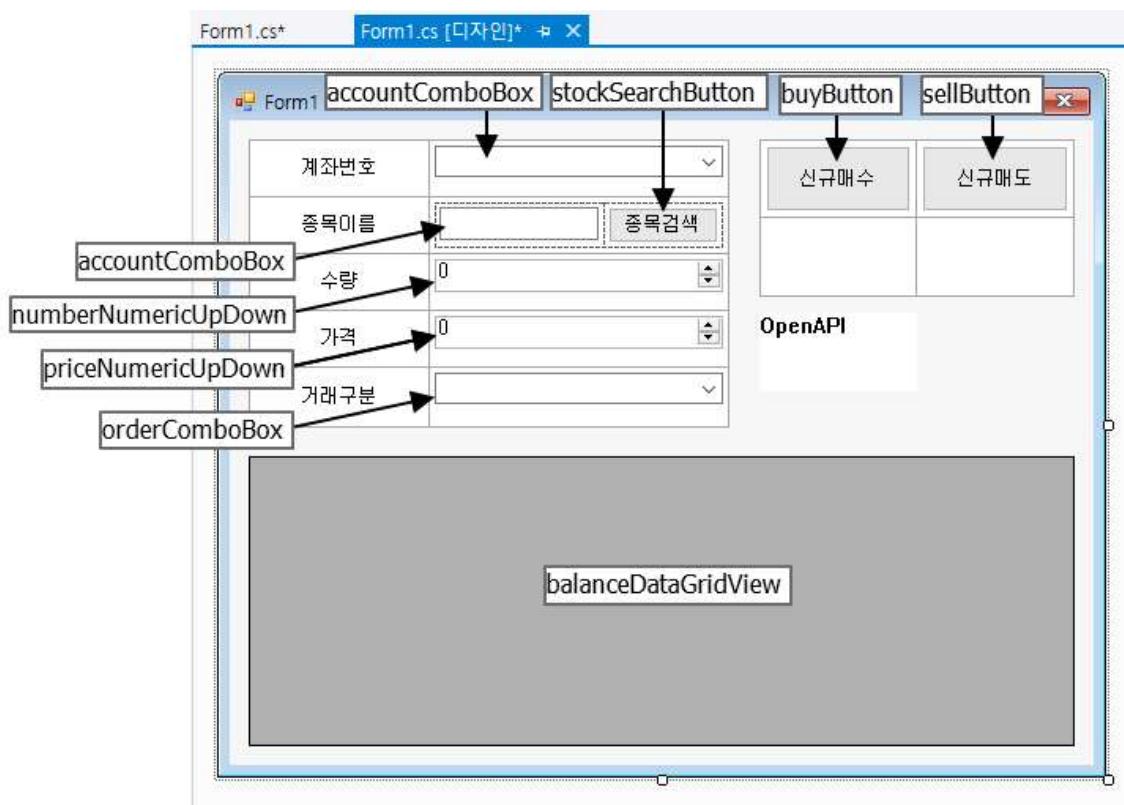
OpenAPI
잔고조회

| 종목코드     | 종목명    | 수량 | 매수금       | 현재가     |
|----------|--------|----|-----------|---------|
| A004105  | 태양금속우  | 10 | 45,150    | 4,445   |
| A007310  | 오뚜기    | 6  | 4,395,000 | 744,000 |
| A007390  | 네이처셀   | 10 | 305,500   | 30,350  |
| A010040  | 한국내화   | 10 | 46,880    | 4,670   |
| A013700  | 까뮤미앤씨  | 10 | 17,000    | 1,810   |
| A039490  | 키움증권   | 2  | 240,000   | 123,500 |
| 10000000 | 테헤란로우드 | 10 | 21,150    | 2,150   |

## 7.2 주식 매도 주문 하기

7.2장에서는 보유하고 있는 종목을 매도하는 방법에 대해서 알아보도록 하겠습니다.

7.1장에서는 보유한 주식 잔고 정보를 요청해서 DataGridView에 바인딩 해봤습니다. 7.2장에서는 DataGridView에 바인딩 된 항목을 선택하고 매도주문을 요청하는 방법을 다릅니다. 먼저 화면을 구성합니다.



7.2장에서는 아래의 순서로 코드를 작성합니다.

- 화면에 계좌번호, 거래구분, 자동완성 목록 바인딩 □
- 잔고조회 Button 클릭하면 balanceDataGridView에 주식 잔고 바인딩□
- balanceDataGridView에서 종목 선택하면 화면에 가격 바인딩□
- 신규매도 Button 코드 작성□

1번부터 코드를 작성해보겠습니다. 1번 항목은 OnEventConnect 함수를 사용해서 화면 컨트롤에 값을 바인딩하면 됩니다. Form1.cs 의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
}
```

함수의 내용을 작성합니다. OnEventConnect 함수에서는 아래 순서대로 화면에 값을 바인딩 합니다.

- "00:지정가" , "03:시장가" 문자열을 화면에 바인딩합니다.
- 계좌번호를 화면에 바인딩합니다.
- 종목 자동완성 목록을 화면에 바인딩합니다.

코드로 작성하면 아래와 같습니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
 }
}
```

- 화면에 계좌번호, 거래구분, 자동완성 목록 바인딩
- 잔고조회 Button 클릭하면 balanceDataGridView에 주식 잔고 바인딩
- balanceDataGridView에서 종목 선택하면 화면에 가격 바인딩
- 신규매도 Button 코드 작성

다음은 잔고조회 Button을 클릭했을때 balanceDataGridView에 주식 잔고가 바인딩 되도록 코드를 작성해보겠습니다. 신규매수 Button, StockSearchButton에 관한 코드는 이전의 6.1장 6.2장을 참고하시면 작성 할 수 있습니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
}
```

balanceCheckButton을 클릭했을때 동작할 코드는 balanceCheck 함수입니다. balanceCheck 함수를 Form1.cs의 생성자에 추가했다면 함수의 내용을 작성합니다.

```
public void balanceCheck(object sender, EventArgs e)
{
 accountNumber = accountComboBox.Text;
 string password = ";//사용자 계좌 비밀번호 입력

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "0004");
}
```

계좌 평가 현황을 요청하고 OnReceiveTrData 함수를 작성해서 주식 잔고를 요청합니다. 7.1장의 내용과 같은 내용입니다. OnReceiveTrData 함수도 Form1.cs의 생성자에 추가하고 작성합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
```

```

 axKHOpenAPI1.OnEventConnect += onEventConnect;

 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
}

```

함수를 추가하고 함수의 내용을 작성합니다.

```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
 else if (e.sRQName == "계좌평가현황요청")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "보유수량"));
 long buyingMoney =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 long currentPrice =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));
 long estimatedProfit =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));
 double estimatedProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalanceList.Add(new stockBalance(stockCode, stockName,
number, String.Format("{0:#,###}", buyingMoney), String.Format("{0:#,###}",


```

```

 currentPrice), estimatedProfit, String.Format("{0:f2}", estimatedProfitRate));
 }
 balanceDataGridView.DataSource = stockBalanceList;
}
}

```

이렇게 코드의 작성을 마쳤다면 7.1장처럼 화면의 balanceDataGridView에 사용자의 주식 잔고 정보가 바인딩 됩니다.

- 화면에 계좌번호, 거래구분, 자동완성 목록 바인딩
- 잔고조회 Button 클릭하면 balanceDataGridView에 주식 잔고 바인딩
- balanceDataGridView에서 종목 선택하면 화면에 가격 바인딩
- 신규매도 Button 코드 작성

다음은 balanceDataGridView에서 종목을 선택했을 때 동작할 코드를 작성하겠습니다.

balanceDataGridView에서 종목을 선택했을 때 함수가 동작하도록 하려면 Form1.cs의 생성자에 함수를 아래와 같이 추가하면 됩니다.

```

public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
 balanceDataGridView.SelectionChanged += bindingByBalanceDataGridView; // 추가한 함수
}

```

DataGridView 내의 선택 변화를 뜻하는 SelectionChanged에 함수를 추가합니다. 함수의 이름은 bindingByBalanceDataGridView라고 지었습니다. 함수의 내용을 작성합니다.

```

public void bindingByBalanceDataGridView(object sender, EventArgs e)
{
 if (balanceDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 string[] stockPriceArray = balanceDataGridView["현재가", rowIndex].Value.ToString().Split(',');
 string stockPrice = "";
 for (int i = 0; i < stockPriceArray.Length; i++)
 }
}

```

```

 {
 stockPrice = stockPrice + stockPriceArray[i];
 }
 long stockCurrentPrice = long.Parse(stockPrice);
 priceNumericUpDown.Value = stockCurrentPrice;
}
}

```

if문을 사용해서 balanceDataGridView에서 선택한 Cell이 있다면 코드가 동작하도록 합니다. 선택된 행의 rowIndex를 가져와서 DataGridView의 값에 접근합니다. DataGridView에 있는 "현재가" 값이 priceNumericUpDown 컨트롤에 바인딩 되는 과정입니다. 이렇게 하면 사용자가 매도 주문을 하기 전에 컨트롤에 값을 편리하게 바인딩 할 수 있습니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 잔고조회 Button 클릭하면 balanceDataGridView에 주식 잔고 바인딩
- balanceDataGridView에서 종목 선택하면 화면에 가격 바인딩
- 신규매도 Button 코드 작성

가격의 바인딩이 완료되었다면 신규매도 Button의 코드를 작성합니다. 6.1장의 매수주문하기에서는 if() 문에서 sender 가 buyButton일때 종목매수를 요청하도록 코드를 작성했었습니다. 7.2장에서는 else if문에서 sender가 sellButton이면 종목 매도를 요청하도록 코드를 작성하겠습니다.

```

public void orderButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

 }
 else if (sender.Equals(sellButton))
 {
 if (balanceDataGridView.SelectedRows.Count > 0)
 {
 string accountCode = accountComboBox.Text;
 int rowIndex =
balanceDataGridView.SelectedCells[0].RowIndex;
 int stockQty = int.Parse(balanceDataGridView["수량",
rowIndex].Value.ToString());
 string stockCode = balanceDataGridView["종목코드",

```

```

rowIndex].Value.ToString().Trim().Replace("A", "");

int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
string[] orderCombo = orderComboBox.Text.Split(':');

axKHOpenAPI1.SendOrder("종목신규매도", "8289", accountCode,
2, stockCode, stockQty, stockPrice, orderCombo[0], "");

}
}
}

```

종목 매도를 요청하기 위해서는 계좌번호 , 종목 수량 , 종목코드 , 종목가격 정보가 필요합니다. 화면의 컨트롤로부터 정보를 가져오고 SendOrder 함수를 호출합니다. 이렇게 코드를 작성하면 신규매도 Button의 코드 작성이 끝난 것입니다.

- 화면에 계좌번호 , 거래구분 , 자동완성 목록 바인딩
- 잔고조회 Button 클릭하면 balanceDataGridView에 주식 잔고 바인딩
- balanceDataGridView에서 종목 선택하면 화면에 가격 바인딩
- 신규매도 Button 코드 작성

아래 코드는 6장의 코드를 포함한 전체 코드입니다.

### 전체 코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp9
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 List<stockBalance> stockBalanceList;
 public string accountNumber = "";
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 }
 }
}

```

```

 buyButton.Click += orderButtonClicked;
 sellButton.Click += orderButtonClicked;
 stockSearchButton.Click += stockSearch;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
 balanceDataGridView.SelectionChanged +=
bindingByBalanceDataGridView;
 }

 public void bindingByBalanceDataGridView(object sender, EventArgs e)
 {
 if (balanceDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 string[] stockPriceArray = balanceDataGridView["현재가",
rowIndex].Value.ToString().Split(',');
 string stockPrice = "";
 for (int i = 0; i < stockPriceArray.Length; i++)
 {
 stockPrice = stockPrice + stockPriceArray[i];
 }
 long stockCurrentPrice = long.Parse(stockPrice);
 priceNumericUpDown.Value = stockCurrentPrice;
 }
 }

 public void balanceCheck(object sender, EventArgs e)
 {
 accountNumber = accountComboBox.Text;
 string password = "://사용자 비밀번호 입력

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "0004");

 }

 public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
 {
 if (e.sRQName == "종목정보요청")
 {

```

```

 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
 else if (e.sRQName == "계좌평가현황요청")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "보유수량"));
 long buyingMoney = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가"
).Replace("-", ""));
 long estimatedProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));
 double estimatedProfitRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익률"));

 stockBalanceList.Add(new stockBalance(stockCode,
stockName, number, String.Format("{0:#,###}", buyingMoney),
String.Format("{0:#,###}", currentPrice), estimatedProfit, String.Format("{0:f2}", estimatedProfitRate)));
 }
 balanceDataGridView.DataSource = stockBalanceList;
 }
}
public void stockSearch(object sender, EventArgs e)
{
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
}

```

```

 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
 }

 public void orderButtonClicked(object sender, EventArgs e)
 {
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length > 0 &&
accountComboBox.Text.Length > 0)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode,
1, orderCode, stockQty, stockPrice, orderCombo[0], "");

 }
 }
 else if (sender.Equals(sellButton))
 {
 if (balanceDataGridView.SelectedRows.Count > 0)
 {
 string accountCode = accountComboBox.Text;
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 int stockQty = int.Parse(balanceDataGridView["수량",
rowIndex].Value.ToString());
 string stockCode = balanceDataGridView["종목코드",
rowIndex].Value.ToString().Trim().Replace("A", "");

 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');
 }
 }
 }
}

```

```

 axKHOpenAPI1.SendOrder("종목신규매도", "8289", accountCode,
2, stockCode, stockQty, stockPrice, orderCombo[0], "");

 }

}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
 }
}

class stockInfo
{
 public string stockCode;
 public string stockName;
}

```

```
public stockInfo(string stockCode, string stockName)
{
 this.stockCode = stockCode;
 this.stockName = stockName;
}
}

class stockBalance
{
 public string 종목코드 { get; set; }
 public string 종목명 { get; set; }
 public long 수량 { get; set; }
 public string 매수금 { get; set; }
 public string 현재가 { get; set; }
 public long 평가손익 { get; set; }
 public string 수익률 { get; set; }

 public stockBalance() { }

 public stockBalance(string 종목번호, string 종목명, long 수량, string 매수
금, string 현재가, long 평가손익, string 수익률)
 {
 this.종목코드 = 종목번호;
 this.종목명 = 종목명;
 this.수량 = 수량;
 this.매수금 = 매수금;
 this.현재가 = 현재가;
 this.평가손익 = 평가손익;
 this.수익률 = 수익률;
 }
}
}
```

실행화면

매도 전 잔고조회 화면(지엔코 2개 보유)

| 증목코드    | 증목명   | 수량 | 매수금     | 현재가     |
|---------|-------|----|---------|---------|
| A007390 | 네이처셀  | 10 | 305,500 | 30,500  |
| A013700 | 까뮤미앤씨 | 10 | 17,000  | 1,725   |
| A039490 | 키움증권  | 2  | 240,000 | 124,000 |
| A065060 | 지엔코   | 2  | 5,790   | 2,875   |
| A224060 | 코디엠   | 10 | 16,250  | 1,625   |
| A252500 | 세화피앤씨 | 10 | 71,100  | 7,110   |

매도 후 잔고조회 화면(지엔코 2개 매도 완료)

Form1

|      |                                                          |
|------|----------------------------------------------------------|
| 계좌번호 | 8101317911                                               |
| 증목이름 | <input type="text"/> <input type="button" value="증목검색"/> |
| 수량   | 2                                                        |
| 가격   | 4465                                                     |
| 거래구분 | 00:지정가                                                   |

신규매수

신규매도

OpenAPI
잔고조회

| 증목코드    | 증목명   | 수량 | 매수금       | 현재가     | 평   |
|---------|-------|----|-----------|---------|-----|
| A004105 | 태양금속우 | 10 | 45,150    | 4,465   | -50 |
| A007310 | 오뚜기   | 6  | 4,395,000 | 745,000 | 750 |
| A007390 | 네이처셀  | 10 | 305,500   | 30,450  | -10 |
| A013700 | 까뮤미앤씨 | 10 | 17,000    | 1,800   | 100 |
| A039490 | 키움증권  | 2  | 240,000   | 123,500 | 700 |
| A224060 | 코디엠   | 10 | 16,250    | 1,630   | 50  |
| A252500 | 세화피앤씨 | 10 | 71,100    | 7,120   | 100 |

< >

## 8. 체결정보

### Preview

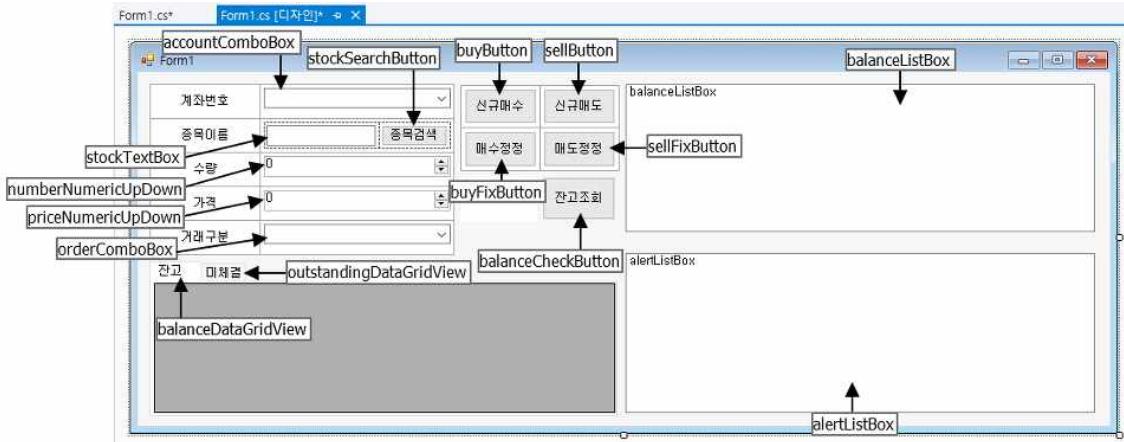
8장에서는 프로그램에서 실시간 주식 주문, 체결 정보를 확인해봅니다. 주식 체결 정보를 확인하는 OnReceiveChejanData 함수를 사용합니다. 미체결 주문도 프로그램에서 확인해봅니다.

### Contents

8.1장에서는 OnReceiveChejanData 함수에 대해서 학습합니다. 사용자의 프로그램에서 주식 주문 체결시 hts처럼 체결 알림을 출력합니다. OnReceiveChejanData 함수에서는 사용자의 주식 잔고, 주문 정보를 모두 확인 할 수 있습니다.

## 8.1 OnReceiveChejanData 함수

8.1장에서는 체결정보를 실시간으로 확인하는 방법에 대해서 알아봅니다. OnReceiveChejanData 함수에 대해서 알아봅니다. 먼저 화면을 구성합니다. 6장과 7장의 화면을 참고하면 조금 더 쉽게 만들 수 있습니다.



balanceDataGridView에는 사용자 보유 주식 잔고를 표시합니다. outstandingDataGridView에는 미체결 주문을 표시합니다. balanceListBox에는 주식 주문 정보, 일별 손익을 표시합니다. alertListBox에는 사용자 주문, 체결 정보를 표시합니다.

### accountComboBox의 속성

| 속성     | 값                        |
|--------|--------------------------|
| (Name) | accountComboBox          |
| Anchor | Top, Bottom, Left, Right |

### stockTextBox의 속성

| 속성                 | 값                        |
|--------------------|--------------------------|
| (Name)             | stockTextBox             |
| AutoCompleteMode   | Suggest                  |
| AutoCompleteSource | CustomSource             |
| Anchor             | Top, Bottom, Left, Right |

### stockSearchButton의 속성

| 속성        | 값                        |
|-----------|--------------------------|
| (Name)    | stockSearchButton        |
| Text      | 종목검색                     |
| TextAlign | MiddleCenter             |
| Anchor    | Top, Bottom, Left, Right |

### numberNumericUpDown의 속성

| 속성      | 값                        |
|---------|--------------------------|
| (Name)  | numberNumericUpDown      |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**priceNumericUpDown의 속성**

|         |                          |
|---------|--------------------------|
| 속성      | 값                        |
| (Name)  | priceNumericUpDown       |
| Maximum | 10000000                 |
| Anchor  | Top, Bottom, Left, Right |

**orderComboBox의 속성**

|        |                          |
|--------|--------------------------|
| 속성     | 값                        |
| (Name) | orderComboBox            |
| Anchor | Top, Bottom, Left, Right |

**balanceDataGridView의 속성**

|                  |                     |
|------------------|---------------------|
| 속성               | 값                   |
| (Name)           | balanceDataGridView |
| RowHeaderVisible | False               |
| SelectionMode    | FullRowSelect       |

**outstandingDataGridView의 속성**

|                  |                         |
|------------------|-------------------------|
| 속성               | 값                       |
| (Name)           | outstandingDataGridView |
| RowHeaderVisible | False                   |
| SelectionMode    | FullRowSelect           |

**buyButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | buyButton                |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매수                     |
| TextAlign | MiddleCenter             |

**sellButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | sellButton               |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 신규매도                     |
| TextAlign | MiddleCenter             |

**buyFixButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | buyFixButton             |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 매수정정                     |
| TextAlign | MiddleCenter             |

**sellFixButton의 속성**

|           |                          |
|-----------|--------------------------|
| 속성        | 값                        |
| (Name)    | sellFixButton            |
| Anchor    | Top, Bottom, Left, Right |
| Text      | 매도정정                     |
| TextAlign | MiddleCenter             |

**balanceListBox의 속성**

|              |                     |
|--------------|---------------------|
| 속성<br>(Name) | 값<br>balanceListBox |
|--------------|---------------------|

**outstandingListBox의 속성**

|              |                         |
|--------------|-------------------------|
| 속성<br>(Name) | 값<br>outstandingListBox |
|--------------|-------------------------|

속성 표를 참고해서 화면을 구성합니다.

먼저 Form1.cs에 로그인이 되었을 때 화면에 필요한 정보를 바인딩 하는 코드를 작성하겠습니다. OnEventConnect 함수를 Form1.cs 의 생성자에 추가하고 함수의 내용을 정의합니다. accountComboBox , orderComboBox 에 계좌번호와 거래구분 정보를 바인딩 합니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 orderComboBox.Items.Add("00:지정가").ToString());
 orderComboBox.Items.Add("03:시장가").ToString());
 }
}
```

먼저 거래 구분 정보를 바인딩합니다. 00:지정가와 03:시장가를 바인딩 하는 이유에 대해서 설명하도록 하겠습니다.

|                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [SendOrder() 함수] 설명                                                                                                                                                                                                                                                                                                                                                   |
| SendOrder(<br>BSTR sRQName, // 사용자 구분명<br>BSTR sScreenNo, // 화면번호<br>BSTR sAccNo, // 계좌번호 10자리<br>LONG nOrderType, // 주문유형 1:신규매수, 2:신규매도 3:매수취소, 4:매도취소,<br>5:매수정정, 6:매도정정<br>BSTR sCode, // 종목코드<br>LONG nQty, // 주문수량<br>LONG nPrice, // 주문가격<br>BSTR sHogaGb, // 거래구분(혹은 호가구분)은 아래 참고<br>BSTR sOrgOrderNo // 원주문번호입니다. 신규주문에는 공백, 정정(취소)주문할<br>원주문번호를 입력합니다.<br>) |

9개 인자값을 가진 국내 주식주문 함수이며 리턴값이 0이면 성공이며 나머지는 에러입니다.

1초에 5회만 주문가능하며 그 이상 주문요청하면 에러 -308을 리턴합니다.

[거래구분]

모의투자에서는 지정가 주문과 시장가 주문만 가능합니다.

00 : 지정가  
 03 : 시장가  
 05 : 조건부지정가  
 06 : 최유리지정가  
 07 : 최우선지정가  
 10 : 지정가IOC  
 13 : 시장가IOC  
 16 : 최유리IOC  
 20 : 지정가FOK  
 23 : 시장가FOK  
 26 : 최유리FOK  
 61 : 장전시간외종가  
 62 : 시간외단일가매매  
 81 : 장후시간외종가

KOA Studio의 SendOrder 함수 설명입니다. SendOrder 함수를 호출해서 매수 주문을 할 때 거래구분에 00, 03, 05 같은 문자를 적게 될 것입니다. ComboBox에 먼저 아래와 같이 문자열을 바인딩 해놓으면

00:지정가 , 03:시장가

SendOrder 함수를 호출하기 전에 :(콜론) 을 기준으로 00:지정가 문자열을 분리해서 00 또는 03을 인자 값으로 적을 수 있습니다. 프로그램 실행 화면에서 BuyButton을 클릭했을 때 사용자가 ComboBox에서 선택한 00또는 03을 가져오도록 코드를 작성하는 과정입니다.

OnEventConnect 함수의 내용을 더 작성합니다. OnEventConnect 함수에서는 총 3가지 정보를 바인딩 해야합니다.

- 계좌번호
- 종목이름 자동완성목록
- 거래구분

00:지정가 , 03:시장가 거래구분을 모두 바인딩 했으니, 다음은 계좌번호를 바인딩하도록 하겠습니다.

```
string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
string[] accountArray = accountList.Split(';');
for (int i = 0; i < accountArray.Length; i++)
{
 accountComboBox.Items.Add(accountArray[i]);
}
```

이전에 3.2장에서 계좌번호를 ComboBox에 바인딩 했던 것처럼 코드를 작성하면 됩니다. 종목 이름 자동완성 목록은 4.1장의 TextBox에 AutoCompleteStringCollection을 추가하는 방법을 참고하면 됩니다.

```
string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(';');
AutoCompleteStringCollection stockCollection = new AutoCompleteStringCollection();

for (int i = 0; i < stockArray.Length; i++)
{
 stockList.Add(new stockInfo(stockArray[i]));
 axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
}
stockTextBox.AutoCompleteCustomSource = stockCollection;
```

매수주문을 실행할 때 수량과 가격은 사용자가 직접 작성하므로 OnEventConnect 함수에서 바인딩 하지 않습니다. OnEventConnect 함수는 결과적으로 아래와 같이 작성이 됩니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 orderComboBox.Items.Add("00:지정가".ToString());
 orderComboBox.Items.Add("03:시장가".ToString());

 stockList = new List<stockInfo>();
 string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
 string[] accountArray = accountList.Split(';');
 for (int i = 0; i < accountArray.Length; i++)
 {
 accountComboBox.Items.Add(accountArray[i]);
 }
 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
```

```

 axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
}
}

```

다음은 사용자 주식 잔고와 미체결 정보를 TrData 함수를 호출해서 바인딩 합니다. 잔고 조회 버튼에 잔고조회 , 미체결 조회를 요청하는 함수를 추가하도록 하겠습니다.

---

OnReceiveChejanData 함수는 SendOrder 함수를 호출했을 때 호출되는 이벤트 함수입니다. OnReceiveChejanData 함수를 사용하면 사용자가 현재 가지고 있는 주식 잔고의 실시간 가격을 알 수 있습니다.

|                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> [OnReceiveChejanData() 이벤트] OnReceiveChejanData(     BSTR sGubun, // 체결구분 접수와 체결시 '0'값, 국내주식 잔고전달은 '1'값, 파     생잔고 전달은 '4'     LONG nItemCnt,     BSTR sFIdList ) </pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

주식 주문 접수 , 체결시 OnReceiveChejanData 함수의 sGubun 값에는 0이 전달됩니다. 국내 주식 잔고 전달은 1이 전달되고, 파생 잔고 전달은 4가 전달됩니다. 이렇게 전달되는 서로 다른 값을 사용해서 함수의 내용을 구현하면 아래 코드처럼 구현하게 됩니다.

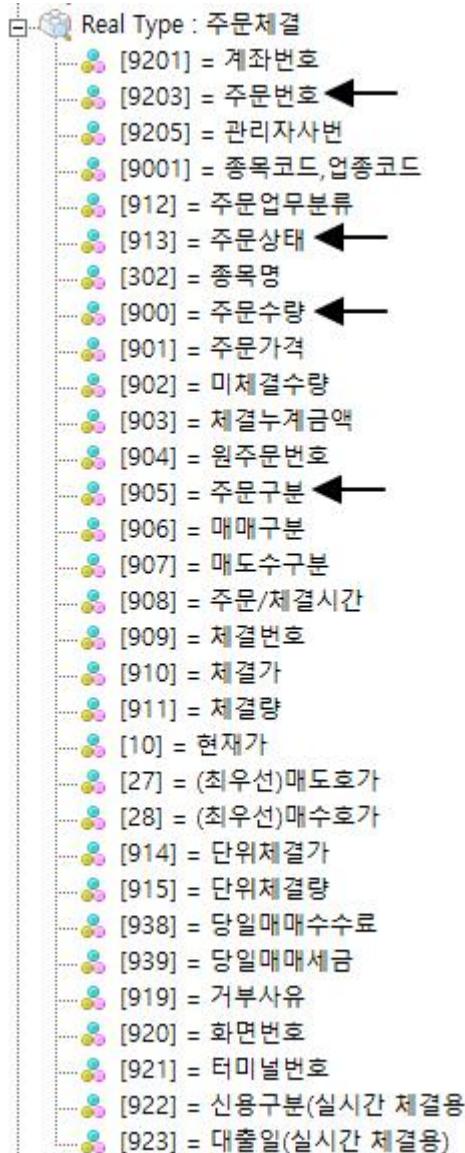
|                                                                                                                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> public void onReceiveChejanData(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveChejanDataEvent e) {     if (e.sGubun=="0")//주문 접수 , 체결시     {         axKHOpenAPI1.GetChejanData(10);     }     else if (e.sGubun=="1")//국내주식 잔고전달     {         string 계좌번호 = axKHOpenAPI1.GetChejanData(9201);     } } </pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

사용자는 OnReceiveChejanData 함수 안에서 GetChejanData 함수를 호출함으로써 실시간 체결정보를 요청 할 수 있습니다.

SendOrder 함수를 사용해서 주식을 매수합니다

OnReceiveChejanData 함수 안에서 GetChejanData 함수를 호출할때 RealType : 주문체결 안에 있는 Fid들을 호출하면 됩니다.

GetChejanData 함수를 사용하면 HTS에서 주식 주문을 하는것처럼 주문이 체결 되었을 때 사용자가 프로그램에서 체결 알림을 받을 수 있습니다.



KOA Studio를 참고해서 함수의 인자 값으로 Fid를 작성하면 주식의 주문/체결이 되었을때 사용자는 계좌번호 , 주문상태 , 주문수량 등의 정보를 요청 할 수 있습니다. "RealType:주문체결" 은 sGubun 값이 0일때 요청하면 됩니다. sGubun 의 값이 1일때는 "RealType : 잔고" 의 Fid를 사용해서 실시간 데이터를 요청 할 수 있습니다.

실현순익화면

[0329] 실현손익 - 일별 실현손익(모의투자)

| 당일실현손익상세                                                                                      |            | 종목별당일손익    |            | 종목별실현손익                             |                          | 일별실현손익 |    |
|-----------------------------------------------------------------------------------------------|------------|------------|------------|-------------------------------------|--------------------------|--------|----|
| 계좌번호                                                                                          | 8101-3179  | 모의_우종선     |            | <input checked="" type="radio"/> 일별 | <input type="radio"/> 월별 | 조회     | 다음 |
| 조회기간                                                                                          | 2018/05/01 | ~          | 2018/05/09 | * 수익률의 경우 2016년 3월부터 제공됩니다.         |                          |        |    |
| * 실현손익, 수수료, 세금은 추정치이며, 수수료는 체결시 수수료율로 적용됩니다.<br>* 매입금액, 매도금액, 수수료, 세금은 당일매매일지 화면의 내용과 동일합니다. |            |            |            |                                     |                          |        |    |
| 총매수                                                                                           | 1,868,195  | 총매도        | 11,194,805 | 실현손익                                | 417,990                  |        |    |
| 수수료                                                                                           | 45,580     | 세금합        | 33,571     | 총수익률                                | 3.92%                    |        |    |
| 매매일                                                                                           | 매수금액       | 매도금액       | 실현손익       | 수익률                                 | 수수료                      | 세금     |    |
| 2018/05/09                                                                                    | 1,868,195  | 11,194,805 | 417,990    | 3.92%                               | 45,580                   | 33,571 |    |

balanceListBox 에는 사용자 주문 접수 , 체결시 실현 손익이 표시됩니다.

### 전체 코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp7
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 List<stockBalance> stockBalanceList;
 List<outstanding> outstandingList;
 public string accountNumber = "";
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 axKHOpenAPI1.OnReceiveChejanData += onReceiveChejanData;
 buyButton.Click += orderButtonClicked;
 stockSearchButton.Click += stockSearch;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 balanceCheckButton.Click += balanceCheck;
 }
 }
}

```

```

public void balanceCheck(object sender, EventArgs e)
{
 accountNumber = accountComboBox.Text;
 string password = "2351";

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "0004");

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("체결구분", "1");//1:미체결조회
 axKHOpenAPI1.SetInputValue("매매구분", "2");//2:매수
 axKHOpenAPI1.CommRqData("실시간미체결요청", "opt10075", 0, "0075");

}

public void onReceiveTrData(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "종목정보요청")
 {
 string stockPrice = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "현재가");
 priceNumericUpDown.Value = long.Parse(stockPrice.Replace("-", ""));
 }
 else if (e.sRQName == "계좌평가현황요청")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "보유수량"));
 long buyingMoney = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 }
 }
}

```

```

 long currentPrice =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));

 long estimatedProfit =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));

 double estimatedProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalance(stockCode, stockName, number, String.Format("{0:#,###}", buyingMoney),
String.Format("{0:#,###}", currentPrice), estimatedProfit, String.Format("{0:f2}",
estimatedProfitRate)));
}

balanceDataGridView.DataSource = stockBalanceList;

}

else if (e.sRQName == "실시간미체결요청")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 outstandingList = new List<outstanding>();
 for (int i = 0; i < count; i++)
 {
 string orderCode =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "주문번호
").ToString());

 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").Trim();

 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();

 int orderNumber =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "주문수량"));

 int orderPrice =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "주문가격"));

 int outstandingNumber =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "미체결수량"));

 int currentPrice =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가
").Replace("-", ""));

 string orderGubun = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문구분").Trim();

 string orderTime = axKHOpenAPI1.GetCommData(e.sTrCode,

```

```

e.sRQName, i, "시간").Trim();

 outstandingList.Add(new outstanding(orderCode, stockCode,
stockName, orderNumber, String.Format("{0:#,###}", orderPrice),
String.Format("{0:#,###}", currentPrice), outstandingNumber, orderGubun,
orderTime));

 }

 outstandingDataGridView.DataSource = outstandingList;
 }
 }

 public void stockSearch(object sender, EventArgs e)
 {
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName==stockText);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");

 }

 public void onReceiveChejanData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveChejanDataEvent e)
 {
 if (e.sGubun=="0")//주문 접수 , 체결시
 {
 alertListBox.Items.Add("계좌번호 : " + "+"
axKHOpenAPI1.GetChejanData(9201)+" | "+ " 주문번호 : " + "+"
axKHOpenAPI1.GetChejanData(9203));

 alertListBox.Items.Add("주문상태 : "
"+axKHOpenAPI1.GetChejanData(913)+" | "+ " 종목명 : " + "+"
axKHOpenAPI1.GetChejanData(302));

 alertListBox.Items.Add("매매구분 : "
+a l e r t L i s t B o x . I t e m s . A d d (" 매 매 구 분
"+axKHOpenAPI1.GetChejanData(906) + " | " + " +" 주문수량 : "
"+axKHOpenAPI1.GetChejanData(900));

 string orderTime = axKHOpenAPI1.GetChejanData(908);
 string orderHour = orderTime[0] +""+ orderTime[1];
 string orderMinute = orderTime[2] + "" + orderTime[3];
 string orderSecond = orderTime[4] + "" + orderTime[5];
 long orderPrice = long.Parse(axKHOpenAPI1.GetChejanData(901));
 }
 }
}

```

```

 alertListBox.Items.Add("주문/체결시간 : " + orderHour+"시"
"+orderMinute+"분 "+orderSecond+"초");
 alertListBox.Items.Add("주문구분 : " + axKHOpenAPI1.GetChejanData(905));
 alertListBox.Items.Add("주문가격 : " + String.Format("{0:#,###}",orderPrice));

alertListBox.Items.Add("-----");
}

else if (e.sGubun=="1")//국내주식 잔고전달
{
 string stockName = axKHOpenAPI1.GetChejanData(302);
 long currentPrice = long.Parse(axKHOpenAPI1.GetChejanData(10).Replace("-", ""));
 string profitRate = axKHOpenAPI1.GetChejanData(8019);
 long totalBuyingPrice = long.Parse(axKHOpenAPI1.GetChejanData(932));
 long profitMoney = long.Parse(axKHOpenAPI1.GetChejanData(950));

 balanceListBox.Items.Add("종목명 : "+stockName+ " | 현재 종가 : "
+String.Format("{0:#,###}",currentPrice));
 balanceListBox.Items.Add("매입주문금액 : " + String.Format("{0:#,###}",totalBuyingPrice) + " | 금일 실현손익 : " +
String.Format("{0:#,###}",profitMoney));
 balanceListBox.Items.Add("금일 실현 손익율 : " + profitRate);

balanceListBox.Items.Add("-----");
}

public void orderButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 if (stockTextBox.Text.Length>0 &&

```

```

accountComboBox.Text.Length>0)
{
 string stockText = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockText);

 string orderCode = stockList[index].stockCode;
 string accountCode = accountComboBox.Text;
 int stockQty = int.Parse(numberNumericUpDown.Value.ToString());
 int stockPrice = int.Parse(priceNumericUpDown.Value.ToString());
 string[] orderCombo = orderComboBox.Text.Split(':');

 axKHOpenAPI1.SendOrder("종목신규매수", "8249", accountCode,
1, orderCode, stockQty, stockPrice, orderCombo[0], "");
}

else if (sender.Equals(sellButton))
{
 if (balanceDataGridView.SelectedCells[0]!=null)
 {
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 Console.WriteLine(rowIndex);

 Console.WriteLine(stockBalanceList[rowIndex].종목코드);

 }
}

//Console.WriteLine(balanceDataGridView["종목코드",
rowIndex].Value);

}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 orderComboBox.Items.Add("00:지정가").ToString());
 }
}

```

```

orderComboBox.Items.Add("03:시장가".ToString());

stockList = new List<stockInfo>();
string accountList = axKHOpenAPI1.GetLoginInfo("ACCLIST");
string[] accountArray = accountList.Split(';');
for (int i = 0; i < accountArray.Length; i++)
{
 accountComboBox.Items.Add(accountArray[i]);
}

string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
string[] stockArray = stocklist.Split(';');
AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

for (int i = 0; i < stockArray.Length; i++)
{
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));
}

stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
}
stockTextBox.AutoCompleteCustomSource = stockCollection;
}

class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode, string stockName)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}

class outstanding
{
 public string 주문번호 { get; set; }
 public string 종목코드 { get; set; }
 public string 종목명 { get; set; }
}

```

```

public int 주문수량 { get; set; }
public string 주문가격 { get; set; }
public int 미체결수량 { get; set; }
public string 주문구분 { get; set; }
public string 현재가 { get; set; }
public string 시간 { get; set; }

public outstanding()
{
}

public outstanding(string 주문번호, string 종목코드, string 종목명, int 주문
수량, string 주문가격, string 현재가, int 미체결수량, string 주문구분, string 시간)
{
 this.주문번호 = 주문번호;
 this.종목코드 = 종목코드;
 this.종목명 = 종목명;
 this.주문수량 = 주문수량;
 this.주문가격 = 주문가격;
 this.미체결수량 = 미체결수량;
 this.주문구분 = 주문구분;
 this.현재가 = 현재가;
 this.시간 = 시간;
}

class stockBalance
{
 public string 종목코드 { get; set; }
 public string 종목명 { get; set; }
 public long 수량 { get; set; }
 public string 매수금 { get; set; }
 public string 현재가 { get; set; }
 public long 평가손익 { get; set; }
 public string 수익률 { get; set; }

 public stockBalance() { }

 public stockBalance(string 종목번호, string 종목명, long 수량, string 매수
금, string 현재가, long 평가손익, string 수익률)
{
}

```

```
this.종목코드 = 종목번호;
this.종목명 = 종목명;
this.수량 = 수량;
this.매수금 = 매수금;
this.현재가 = 현재가;
this.평가손익 = 평가손익;
this.수익률 = 수익률;
```

## 실행화면

Form1

|      |            |
|------|------------|
| 계좌번호 | 8101317911 |
| 증목이름 | 오뚜기        |
| 수량   | 3          |
| 가격   | 733000     |
| 거래구분 | 00:지정가     |

|         |      |
|---------|------|
| 신규매수    | 신규매도 |
| 매수정정    | 매도정정 |
| OpenAPI |      |
| 잔고조회    |      |

|           |                   |
|-----------|-------------------|
| 증명 : 삼성전자 | 현재 증가 : 51,000    |
| 증명 : 삼성전자 | 금일 실현손익 : 255,000 |
| 증명 : 삼성전자 | 현재 증가 : 51,100    |
| 증명 : 삼성전자 | 금일 실현손익 : 204,000 |
| 증명 : 삼성전자 | 현재 증가 : 51,100    |
| 증명 : 삼성전자 | 금일 실현손익 : -402    |
| 증명 : 삼성전자 | 현재 증가 : 51,100    |
| 증명 : 삼성전자 | 금일 실현손익 : -79     |
| 증명 : 삼성전자 | 현재 증가 : 51,100    |
| 증명 : 삼성전자 | 금일 실현손익 : -2,044  |
| 증명 : 삼성전자 | 현재 증가 : 51,100    |
| 증명 : 삼성전자 | 금일 실현손익 : -860    |

|                    |                |
|--------------------|----------------|
| 주문상태 : 개설          | 증명별 : 삼성전자     |
| 주문번호 : 8101317911  | 증명수 : 5        |
| 체결시간 : 12시 53분 41초 |                |
| 주문구분 : 정도          |                |
| 주가 : 51,100        |                |
| 계좌번호 : 8101317911  | 주문번호 : 0061049 |
| 증명 : 오뚜기           | 증명 : 오뚜기       |
| 수량 : 3             | 수량 : 3         |
| 매수금 : 153,000      | 매수금 : 153,000  |
| 현재가 : 51,100       | 현재가 : 51,100   |

Form1

|      |            |
|------|------------|
| 계좌번호 | 8101317911 |
| 종목이름 | 오뚜기        |
| 수량   | 3          |
| 가격   | 733000     |
| 거래구분 | 00:지정가     |

**OpenAPI**

|      |      |
|------|------|
| 신규매수 | 신규매도 |
| 매수정정 | 매도정정 |
| 간고조회 |      |

|                                         |                |
|-----------------------------------------|----------------|
| 증정 전자<br>금액 : 255,000   금일 실현순익 : 0,00  | 현재 증가 : 51,000 |
| 상정 전자<br>금액 : 204,000   금일 실현순익 : -402  | 현재 증가 : 51,100 |
| 상정 전자<br>금액 : 204,044   금일 실현순익 : -0,79 | 현재 증가 : 51,100 |
| 상정 전자<br>금액 : 204,044   금일 실현순익 : -0,80 | 현재 증가 : 51,100 |

| 주문번호  | 증목코드   | 증목명   | 주문수량 | 주문가격    |
|-------|--------|-------|------|---------|
| 57741 | 007310 | 오뚜기   | 5    | 714,000 |
| 57710 | 068270 | 셀트리온  | 5    | 246,000 |
| 57645 | 033170 | 시그네틱스 | 5    | 1,100   |

## 9. 조건식

### Preview

9장에서는 프로그램에 사용자의 조건식을 불러옵니다. 조건식을 활용해서 종목을 검색해봅니다. 실시간으로 조건식에 편입되는 종목과 이탈되는 종목도 조회해봅니다.

### Contents

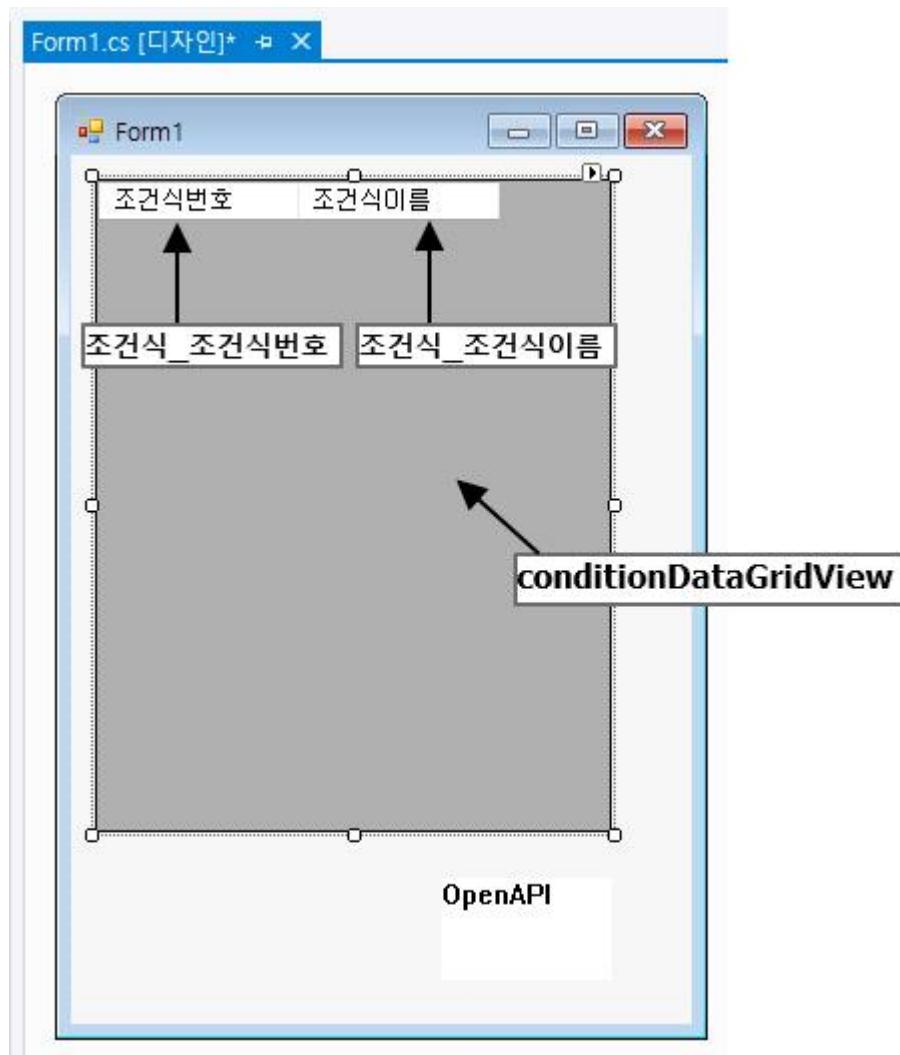
9.1장에서는 hts에서 확인 할 수 있는 사용자의 조건식을 프로그램에서 확인해봅니다. GetConditionLoad 함수를 사용해서 사용자의 조건식을 호출합니다. OnReceiveConditionVer 함수에 대해서도 학습해봅니다.

9.2장에서는 조건식을 사용해서 종목을 검색해봅니다. SendCondition 함수를 사용해서 조건식으로 종목을 검색합니다. OnReceiveTrCondition 함수에 대해서 알아봅니다.

9.3장에서는 실시간으로 조건식에 편입, 이탈되는 종목을 확인합니다. OnReceiveRealCondition 함수에 대해서 알아봅니다.

## 9.1 사용자 조건식 불러오기

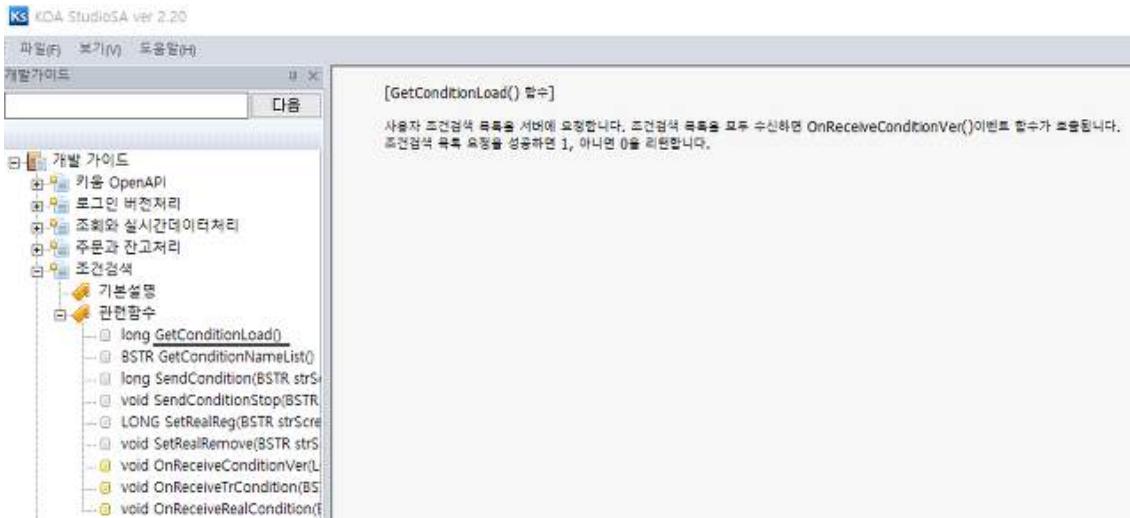
9.1장은 사용자 조건식을 불러오는 방법입니다. 먼저 화면을 구성합니다.



conditionDataGridView의 속성

| 속성<br>(Name)          | 값             |
|-----------------------|---------------|
| AllowUserToAddRows    | False         |
| AllowUserToDeleteRows | False         |
| RowHeaderVisible      | False         |
| SelectionMode         | FullRowSelect |

화면 구성을 마쳤다면 Form1.cs에 코드를 작성해서 조건식을 요청하겠습니다. 조건식 요청하는 순서는 다음과 같습니다.



**GetConditionLoad(일반함수) 호출 -> OnReceiveConditionVer(이벤트 함수) 호출 -> GetConditionNameList(일반함수) 호출**

GetConditionLoad 함수를 호출하면 OnReceiveConditionVer이벤트 함수가 호출됩니다. OnReceiveConditionVer함수 안에서 GetConditionNameList 함수를 호출해서 조건식 목록을 가져옵니다.

9.1장에서는 사용자가 로그인하면 호출되는 OnEventConnect 함수 안에서 GetConditionLoad함수를 호출하도록 하겠습니다.

```
public void onEventConnect(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 axKHOpenAPI1.GetConditionLoad();
 }
}
```

axKHOpenAPI1.GetConditionLoad(); 함수는 조건 검색 목록 요청을 성공하면 1, 아니면 0을 리턴합니다. OnReceiveConnect 함수에서 GetConditionLoad 함수를 호출했다면 이제 Form1.cs의 생성자에 OnReceiveConditionVer함수를 추가합니다.

```
axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
```

OnReceiveConditionVer 함수의 내용을 작성합니다. OnReceiveConditionVer 함수의 AxKHOpenAPILib.\_DKHOpenAPIEventsOnReceiveConditionVerEvent 타입 매개변수에는 사용자가 GetConditionLoad를 통해서 보낸 요청의 결과가 담겨 있습니다.

```
public void onReceiveConditionVer(object sender, AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 if (e.lRet == 1)
```

```
{
 axKHOpenAPI1.GetConditionNameList();
}
}
```

함수를 정의한 모습입니다. 객체 e에 담긴 IRet 결과값이 1이면 정상적으로 조건식 정보가 요청 된 것입니다. 조건식 정보가 잘 요청 되었다면 GetConditionNameList 함수를 호출해서 조건식 이름들을 요청합니다. GetConditionNameList 함수를 호출해서 조건식을 요청하면 아래와 같은 형태로 결과가 반환됩니다.

007^조건식1;008^조건식2;014^조건식3;015^조건식4;020^조건식5;000^조건식6;  
; 과 ^로 구분되어 있는 조건식인덱스와 조건식 이름을 분리해서 화면에 바인딩 하도록 하겠습니다.

맨 끝에 붙어있는 ;은 Split을 사용해서 문자열을 분리 할 때 문제를 발생시키므로 제거해줍니다.

```
string conditionList = axKHOpenAPI1.GetConditionNameList().TrimEnd('');
```

TrimEnd 함수를 사용해서 맨 끝의 ; 을 제거했습니다. 조건식 인덱스와 조건식 이름을 얻기 위해서는 1차적으로 ;을 기준으로 문자열을 분리하고 , 2차로 ^을 기준으로 문자열을 분리해야 합니다. 문자열 맨 끝에 ;이 있으면 ;을 기준으로 분리할때 프로그램은 ; 뒤에 다른 문자열이 더 있는 것으로 인식하기 때문에 에러가 발생합니다. 아래와 같이 코드를 작성하면 조건식 인덱스와 조건식이름을 콘솔창에서 확인 할 수 있습니다.

```
public void onReceiveConditionVer(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 if (e.lRet == 1)
 {
 string conditionList = axKHOpenAPI1.GetConditionNameList().TrimEnd(';');
 string[] conditionArray = conditionList.Split(';');

 for (int i = 0; i < conditionArray.Length; i++)
 {
 string[] condition = conditionArray[i].Split('^');
 Console.WriteLine("인덱스 = " + condition[0]);
 Console.WriteLine("조건식이름 = " + condition[1]);
 }
 }
}
```

for 문 이전에 ;을 사용해서 문자열을 먼저 분리합니다. 그리고 ^를 기준으로 각각의 "조건식인덱스^조건식이름" 문자열을 분리합니다.

이제 분리한 조건식 인덱스와 조건식 이름을 화면에 바인딩 해보도록 하겠습니다.

```
conditionDataGridView.Rows.Add();
conditionDataGridView["조건식_조건식번호", i].Value = condition[0];
conditionDataGridView["조건식_조건식이름", i].Value = condition[1];
```

conditionDataGridView.Rows.Add():conditionDataGridView["조건식\_조건식번호", i].Value = condition[0]:conditionDataGridView["조건식\_조건식이름", i].Value = condition[1];  
DataGridView에 행을 추가하고 인덱스마다 값을 바인딩 합니다.

전체 코드입니다.

```
using System.Windows.Forms;

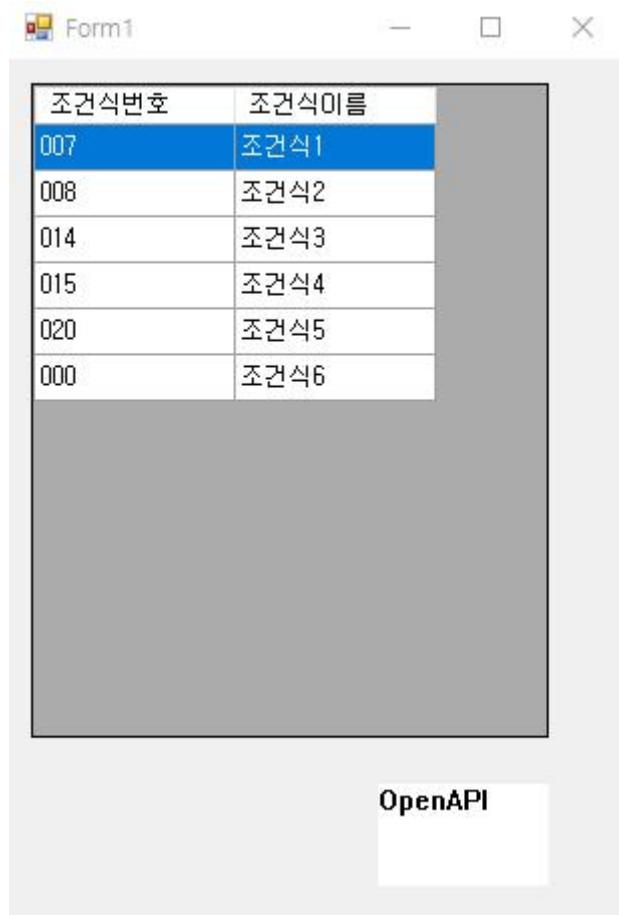
namespace WindowsFormsApp5
{
 public partial class Form1 : Form
 {
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
 }

 public void onReceiveConditionVer(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
 {
 if (e.lRet == 1)
 {
 string conditionList = axKHOpenAPI1.GetConditionNameList().TrimEnd(';');
 string[] conditionArray = conditionList.Split(';');

 for (int i = 0; i < conditionArray.Length; i++)
 {
 string[] condition = conditionArray[i].Split('^');
 conditionDataGridView.Rows.Add();
 conditionDataGridView["조건식_조건식번호", i].Value = condition[0];
 conditionDataGridView["조건식_조건식이름", i].Value = condition[1];
 }
 }
 }
}
```

```
public void onEventConnect(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 axKHOpenAPI1.GetConditionLoad();
 }
}
```

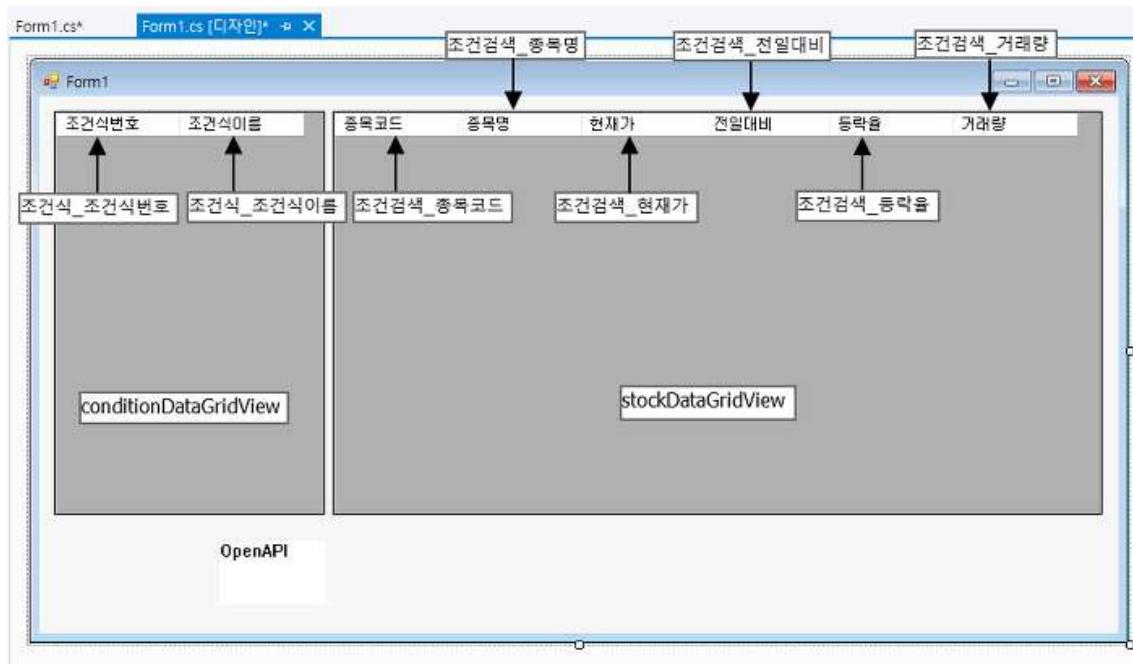
실행화면입니다.



9.2장에서는 조건식을 사용해서 종목을 검색하는 방법에 대해서 알아보도록 하겠습니다.

## 9.2 조건식기반 종목 검색

9.2장에서는 9.1장에서 불러온 조건식을 사용해서 종목을 검색하는 방법을 다룹니다. 9.1장에서 만들었던 화면에 종목들을 표시 할 DataGridView를 추가합니다.



### conditionDataGridView의 속성

| 속성                    | 값                     |
|-----------------------|-----------------------|
| (Name)                | conditionDataGridView |
| AllowUserToAddRows    | False                 |
| AllowUserToDeleteRows | False                 |
| RowHeaderVisible      | False                 |
| SelectionMode         | FullRowSelect         |

### stockDataGridView의 속성

| 속성                    | 값                 |
|-----------------------|-------------------|
| (Name)                | stockDataGridView |
| AllowUserToAddRows    | False             |
| AllowUserToDeleteRows | False             |
| RowHeaderVisible      | False             |
| SelectionMode         | FullRowSelect     |

그림에 DataGridView 안의 열의 이름들도 표시되어 있습니다. 9.2장에서는 왼쪽의 조건식을 클릭하면 오른쪽 DataGridView에 선택한 조건식에 해당하는 종목이 나타나도록 코드를 작성해보겠습니다.

로그인이 되었을 때 conditionDataGridView에 조건식이 나타나도록 OnEventConnect 함수에 조건식을 요청하는 함수를 작성하도록 하겠습니다.

```
if (e.nErrCode==0)
{
 axKHOpenAPI1.GetConditionLoad();
```

```
}
```

GetConditionLoad 함수를 호출하면 OnReceiveConditionVer 이벤트 함수가 호출됩니다. OnReceiveConditionVer 함수 안에서는 GetConditionNameList 함수를 호출해서 사용자가 가지고 있는 조건식 목록을 요청 할 수 있습니다. OnReceiveConditionVer 함수를 사용하기 전에 함수의 설명을 한번 보도록 하겠습니다.

```
KOA Studio
[OnReceiveConditionVer() 이벤트] 함수 설명
OnReceiveConditionVer(
 LONG lRet, // 호출 성공여부, 1: 성공, 나머지 실패
 BSTR sMsg // 호출결과 메시지
)
```

사용자 조건식요청에 대한 응답을 서버에서 수신하면 호출되는 이벤트 함수입니다.

KOA Studio의 OnReceiveConditionVer 함수 설명을 보면 사용자 조건식 호출에 성공하면 변수 lRet에 1을 반환하고, 호출에 실패하면 1이 아닌 값을 반환합니다. lRet의 값이 1인지 먼저 검사하고, 1이 반환되었다면 GetConditionNameList 함수를 호출하면 되겠습니다

Form1.cs의 생성자에 함수를 추가하고, 함수의 내용을 작성합니다.

```
public Form1()
{
 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
}
```

```
public void onReceiveConditionVer(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 if (e.lRet == 1)
 {
 string conditionList = axKHOpenAPI1.GetConditionNameList();
 }
}
```

GetConditionNameList 함수를 호출해서 조건식을 요청하면 조건식이 ; 으로 구분된 문자열로 반환됩니다.

007^조건식1:008^조건식2:014^조건식3:015^조건식4:020^조건식5:000^조건식6;

이 때 문자열의 맨 뒤에 있는 ;은 문자열을 분리할때 문제를 일으킵니다. Split 함수를 사용해서 문자열을 분리할때 ;이 문자열의 맨 뒤에 붙어있으면 컴퓨터는 ;의 뒤에도 문자열이 있는것으로 취급하고 문자열을 분리합니다. 하지만 요청의 결과로 받은 문자열은 맨 뒤의 ; 뒤에 어떠한 문자열도 가지고 있지 않습니다. 그래서 TrimEnd(';) 함수를 사용해서 조건식 문자열 맨 뒤에 있는 ;(세미콜론)을 제거해줍니다.

```
string conditionList = axKHOpenAPI1.GetConditionNameList().TrimEnd('');
```

먼저 ;(세미콜론)을 기준으로 문자열을 분리하고, ^을 기준으로 문자열을 분리해서 화면에 바인딩합니다. 문자열을 분리하기 위해서 Split() 함수를 사용합니다. 분리한 문자열은 string 배열에 저장합니다.

```
string[] conditionArray = conditionList.Split(':'');
```

string 배열 conditionArray에는 ^로 구분된 조건식 문자열들이 들어있습니다.  
for문을 사용해서 conditionArray의 조건식을 모두 분리합니다.

```
for (int i = 0; i < conditionArray.Length; i++)
{
 string[] condition = conditionArray[i].Split('^');
 conditionDataGridView.Rows.Add();
 conditionDataGridView["조건식_조건식번호", i].Value = condition[0];
 conditionDataGridView["조건식_조건식이름", i].Value = condition[1];
}
```

코드를 한 줄씩 살펴보도록 하겠습니다.

```
string[] condition = conditionArray[i].Split('^');
```

conditionArray[0], conditionArray[1] .. conditionArray[] 배열에는 ^로 구분된 조건식 문자열이 들어있습니다. 이것을 ^로 분리해서 string 배열 condition에 저장합니다. 이렇게 문자열을 분리해서 저장하면 condition[0]에는 조건식의 인덱스가 저장됩니다. condition[1]에는 조건식의 이름이 저장됩니다. condition 배열에 조건식을 저장했다면 conditionDataGridView에 바인딩합니다. 화면에 바인딩하기 위한 절차는 아래와 같습니다.

- DataGridView에 새로운 행 추가
- 추가된 행에 조건식 번호 바인딩
- 추가된 행에 조건식 이름 바인딩

DataGridView에 새로운 행을 추가하려면 Rows.Add() 함수를 호출하면 됩니다.

```
conditionDataGridView.Rows.Add();
```

conditionDataGridView에 비어있는 행을 추가하는 함수입니다.

conditionDataGridView에 행을 추가한 후 추가된 행에 조건식 번호를 바인딩합니다. 바인딩하는 방법은 아래와 같습니다.

```
conditionDataGridView["데이터 컬럼의 이름", 행의 위치].Value = 바인딩하려는 값;
```

conditionDataGridView[] 대괄호 안에 데이터 컬럼의 이름과 행의 번호를 적습니다. 이렇게 하면 행 번호와 열의 이름을 가지고 정확한 Cell의 위치를 알 수 있습니다. 그리고 .Value를 사용해서 값을 대입합니다. 9.2장에서는 condition 배열에 있는 값을 하나씩 대입하고 있습니다. condition[0]에는 조건식의 인덱스가 들어있고, condition[1]에는 조건식의 이름이 저장되어 있습니다.

conditionDataGridView에 조건식을 모두 바인딩 했다면 조건식 검색요청을 보내는 코드를 작성합니다. conditionDataGridView에서 사용자의 선택 변화가 일어날때 검색 요청을 보내면 됩니다. 사용자의 선택이 변경되는 것은 SelectionChanged를 사용해서 구현합니다. Form1.cs의 생성자에 SelectionChanged를 추가하고 DataGridView에서 선택 변화가 일어났을때 동작할 함수를 추가합니다.

```
conditionDataGridView.SelectionChanged += conditionSearch;
```

conditionSearch 함수의 내용을 정의합니다.

```
public void conditionSearch(object sender, EventArgs e)
{
}
```

조건식 검색 요청을 보내는 함수는 SendCondition입니다.

```
[SendCondition() 함수]
SendCondition(
 BSTR strScrNo, // 화면번호
 BSTR strConditionName, // 조건식 이름
 int nIndex, // 조건명 인덱스
 int nSearch // 조회구분, 0:조건검색, 1:실시간 조건검색
)
```

SendCondition 함수는 화면번호, 조건식이름, 조건명인덱스, 조회구분을 인자로 적어서 호출하는 함수입니다. 조건식 이름과 인덱스를 적기 위해선 화면에서 선택한 조건식의 인덱스와 이름을 가져오는 방법을 알아야 합니다.

```
int rowIndex = conditionDataGridView.SelectedCells[0].RowIndex;
```

conditionDataGridView.SelectedCells[0].RowIndex를 사용해서 선택한 데이터 행의 행 번호를 알 수 있습니다. SelectedCells[0]은 선택한 셀의 0번째를 의미합니다. conditionDataGridView.SelectedCells[0]을 콘솔에 출력하면 아래와 같은 결과가 출력됩니다.

- DataGridViewTextBoxCell { ColumnIndex=0, RowIndex=0 }
- DataGridViewTextBoxCell { ColumnIndex=0, RowIndex=1 }

DataGridView에서 선택된 열의 번호와 행의 번호를 알려줍니다. conditionDataGridView.SelectedCells의 RowIndex를 호출해서 선택된 셀의 RowIndex를 알아내는 것입니다. RowIndex를 사용해서 조건식의 인덱스, 이름 값을 얻도록 하겠습니다.

```
conditionDataGridView["조건식_조건식이름", rowIndex].Value;
```

Value를 사용해서 접근하면 됩니다. 하지만 Value는 object 타입이고, SendCondition 함수에 인자로 적는 항목은 string입니다. Value를 string으로 바꿔주기 위해서 ToString 메소드를 사용합니다. 프로그램 내에서는 데이터 타입 변환시 에러가 발생하기 쉽습니다. ToString으로 바꾸면서 예외가 발생 할 수 있기 때문에 try-catch 문을 사용해서 해당 구문을 예외처리 합니다.

```

try
{
 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string conditionName = conditionDataGridView["조건식_조건식이름",
rowIndex].Value.ToString();
 }
}
catch (Exception exception)
{
 Console.WriteLine(exception.Message.ToString());
}

```

조건식을 사용해서 종목을 검색할때는 SendCondition 함수를 호출하게 됩니다. SendCondition 함수를 호출하기 위해서는 조건식 이름과 조건식 인덱스가 모두 필요합니다. 화면에서 선택한 조건식 인덱스를 알아내기 위해서 아래 코드처럼 코드를 작성합니다.

```

int conditionIndex = int.Parse(conditionDataGridView["조건식_조건식번호",
",rowIndex].Value.ToString());
axKHOpenAPI1.SendCondition("0156",conditionName,conditionIndex,0);

```

선택한 셀의 값인 Value는 object 타입이므로 ToString 함수를 사용해서 문자열로 변경합니다. 문자열로 변경된 인덱스는 int.Parse 를 사용해서 숫자로 변경 될 수 있습니다. 알아낸 조건식 인덱스와 조건식 이름을 사용해서 SendCondition 함수를 호출합니다. KOA Studio에서 제공하는 SendCondition 함수에 대한 설명은 아래와 같습니다.

[SendCondition() 함수]

```

SendCondition(
BSTR strScrNo, // 화면번호
BSTR strConditionName, // 조건식 이름
int nIndex, // 조건명 인덱스
int nSearch // 조회구분, 0:조건검색, 1:실시간 조건검색
)

```

서버에 조건검색을 요청하는 함수로 맨 마지막 인자값으로 조건검색만 할것인지 실시간 조건검색도 할 것인지를 지정할 수 있습니다.

여기서 조건식 인덱스는 GetConditionNameList()함수가 조건식 이름과 함께 전달한 조건명 인덱스를 그대로 사용해야 합니다.

리턴값 1이면 성공이며, 0이면 실패입니다.

요청한 조건식이 없거나 조건명 인덱스와 조건식이 서로 안맞거나 조회횟수를 초과하는 경우 실패하게 됩니다.

---



---

[조건검색 사용예시]

GetConditionNameList()함수로 얻은 조건식 목록이 "0^조건식1;3^조건식1:8^조건식3:23^조건식5"일때 조건식3을 검색

```
long lRet = SendCondition("0156", "조건식3", 8, 1);
```

화면번호 , 조건식이름 , 조건식인덱스 , 조회구분 을 인자로 입력해서 함수를 호출합니다.

조건식 선택(conditionSearch) 함수에서 SendCondition 함수호출 ->  
OnReceiveTrCondition 함수에서 CommKwRqData 함수호출 -> OnReceiveTrData  
함수에서 종목정보요청 -> 요청결과 화면에 바인딩

//-1로 지정하면 선택된 셀이 없을때 아래에서 -1보다 클때 item을 추가하라고 명시한 조건문에서 걸릴 수 있다. //셀이 선택되지 않았을때 에러를 방지한다.

## 전체 코드

```
using System.Windows.Forms;
using System.Collections.Generic;
using System;

namespace WindowsFormsApp5
{
 public partial class Form1 : Form
 {
 List<ConditionObject> conditionObjectList;
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
 conditionDataGridView.SelectionChanged += conditionSearch;
 axKHOpenAPI1.OnReceiveTrCondition += onReceiveTrCondition;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 }

 public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
 {
 if (e.sRQName == "조건검색종목")
 {
```

```

int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);

int conditionObjectIndex = -1;

if (conditionDataGridView.SelectedCells.Count > 0)
{
 int conditionIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string selectedConditionName = conditionDataGridView["조건식_조건식이름", conditionIndex].Value.ToString();
 conditionObjectIndex = conditionObjectList.FindIndex(o => o.conditionName == selectedConditionName);
}

conditionObjectList[conditionObjectIndex].stockItemList.Clear();
stockDataGridView.Rows.Clear();
for (int i = 0; i < count; i++)
{
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목명").Trim();
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가").Replace("-", ""));
 double upDownRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "등락율"));
 int netChange = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "전일대비"));
 long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "거래량"));

 stockDataGridView.Rows.Add();
 stockDataGridView["조건검색_종목코드", i].Value = stockCode;
 stockDataGridView["조건검색_종목명", i].Value = stockName;
 stockDataGridView["조건검색_현재가", i].Value = String.Format("{0:#,###}", currentPrice);
 stockDataGridView["조건검색_전일대비", i].Value = netChange;
 stockDataGridView["조건검색_등락율", i].Value = upDownRate;
 stockDataGridView["조건검색_거래량", i].Value =
}

```

```

String.Format("{0:#,###}", volume);

 if (conditionObjectIndex>-1)
 {

conditionObjectList[conditionObjectIndex].stockItemList.Add(new
StockItem(stockCode,stockName,currentPrice,upDownRate,netChange,volume));
 }
 }
}

public void onReceiveTrCondition(object sender,
AxKHOpenAPIlib._DKHOpenAPIEvents_OnReceiveTrConditionEvent e)
{
 if (e.strCodeList.Length>0)
 {
 string stockCodeList =
e.strCodeList.Remove(e.strCodeList.Length-1);
 int stockCount = stockCodeList.Split(';').Length;

 if (stockCount <= 100)
 {
 axKHOpenAPI1.CommKwRqData(stockCodeList,0,stockCount,0,"
조건검색종목","5100");
 }
 }
 else if (e.strCodeList.Length==0)
 {
 MessageBox.Show("검색된 종목이 없습니다.");
 }
}

public void conditionSearch(object sender , EventArgs e)
{

try
{
 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex =
conditionDataGridView.SelectedCells[0].RowIndex;
 string conditionName = conditionDataGridView["조건식_조건식"

```

```

이름", rowIndex].Value.ToString();
 int conditionIndex = int.Parse(conditionDataGridView["조건식_"
조건식번호", rowIndex].Value.ToString());

axKHOpenAPI1.SendCondition("0156", conditionName, conditionIndex, 0);
 }

}

catch (Exception exception)
{
 Console.WriteLine(exception.Message.ToString());
}

}

public void onReceiveConditionVer(object sender, ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 conditionObjectList = new List<ConditionObject>();
 if (e.lRet == 1)
 {
 string conditionList =
axKHOpenAPI1.GetConditionNameList().TrimEnd(';');
 string[] conditionArray = conditionList.Split(';');

 for (int i = 0; i < conditionArray.Length; i++)
 {
 string[] condition = conditionArray[i].Split('^');
 conditionDataGridView.Rows.Add();
 conditionDataGridView["조건식_조건식번호", i].Value =
condition[0];
 conditionDataGridView["조건식_조건식이름", i].Value =
condition[1];

 conditionObjectList.Add(new ConditionObject(condition[0],
condition[1]));
 }
 }
}

public void onEventConnect(object sender, ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
}

```

```

 if (e.nErrCode==0)
 {
 axKHOpenAPI1.GetConditionLoad();
 }
 }

 class ConditionObject
 {
 public string conditionIndex;
 public string conditionName;
 public List<StockItem> stockItemList;
 public ConditionObject(string conditionIndex , string conditionName)
 {
 this.conditionIndex = conditionIndex;
 this.conditionName = conditionName;
 stockItemList = new List<StockItem>();
 }
 }

 class StockItem
 {
 public string stockCode;
 public string stockName;
 public long currentPrice;
 public double upDownRate;
 public int netChange;
 public long volume;
 public StockItem(string stockCode , string stockName,long currentPrice,double upDownRate,int netChange,long volume)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 this.currentPrice = currentPrice;
 this.upDownRate = upDownRate;
 this.netChange = netChange;
 this.volume = volume;
 }
 }
}

```

## 실행화면

Form1

| 조건식번호 | 조건식이름 |
|-------|-------|
| 007   | 조건식1  |
| 008   | 조건식2  |
| 014   | 조건식3  |
| 015   | 조건식4  |
| 020   | 조건식5  |
| 000   | 조건식6  |

| 종목코드   | 종목명       | 현재가       | 전일대비  | 등락률   | 거래량       |
|--------|-----------|-----------|-------|-------|-----------|
| 000240 | 한국타이어월... | 18,300    | -50   | -0.27 | 24,045    |
| 000660 | SK하이닉스    | 86,400    | 600   | 0.7   | 1,329,903 |
| 000990 | DB하이텍     | 14,650    | 200   | 1.38  | 147,609   |
| 002350 | 넥센타이어     | 12,700    | 0     | 0     | 42,652    |
| 003240 | 태광산업      | 1,430,000 | 19000 | 1.35  | 593       |
| 003410 | 쌍용양회      | 33,400    | 750   | 2.3   | 432,485   |
| 003550 | LG        | 80,200    | -900  | -1.11 | 93,484    |
| 004020 | 현대제철      | 65,500    | -200  | -0.3  | 883,268   |
| 005490 | POSCO     | 363,000   | 3500  | 0.97  | 81,535    |
| 005930 | 삼성전자      | 51,700    | 100   | 0.19  | 6,307,774 |
| 006650 | 대한유화      | 260,000   | 1000  | 0.39  | 22,407    |
| 007700 | F&F       | 44,550    | 450   | 1.02  | 38,199    |
| 009830 | 한화케미칼     | 29,050    | 50    | 0.17  | 555,446   |

X

검색된 종목이 없습니다.

확인

OpenAPI

Form1

| 조건식번호 | 조건식이름 |
|-------|-------|
| 007   | 조건식1  |
| 008   | 조건식2  |
| 014   | 조건식3  |
| 015   | 조건식4  |
| 020   | 조건식5  |
| 000   | 조건식6  |

| 종목코드   | 종목명       | 현재가       | 전일대비  | 등락률   | 거래량       |
|--------|-----------|-----------|-------|-------|-----------|
| 000240 | 한국타이어월... | 18,300    | -50   | -0.27 | 24,045    |
| 000660 | SK하이닉스    | 86,400    | 600   | 0.7   | 1,329,903 |
| 000990 | DB하이텍     | 14,650    | 200   | 1.38  | 147,609   |
| 002350 | 넥센타이어     | 12,700    | 0     | 0     | 42,652    |
| 003240 | 태광산업      | 1,430,000 | 19000 | 1.35  | 593       |
| 003410 | 쌍용양회      | 33,400    | 750   | 2.3   | 432,485   |
| 003550 | LG        | 80,200    | -900  | -1.11 | 93,484    |
| 004020 | 현대제철      | 65,500    | -200  | -0.3  | 883,268   |
| 005490 | POSCO     | 363,000   | 3500  | 0.97  | 81,535    |
| 005930 | 삼성전자      | 51,700    | 100   | 0.19  | 6,307,774 |
| 006650 | 대한유화      | 260,000   | 1000  | 0.39  | 22,407    |
| 007700 | F&F       | 44,550    | 450   | 1.02  | 38,199    |
| 009830 | 한화케미칼     | 29,050    | 50    | 0.17  | 555,446   |

X

OpenAPI

### 9.3 실시간 조건식 편입 이탈 종목 검색

9.3장에서는 실시간으로 조건식에 편입, 이탈되는 종목들의 정보를 화면에 바인딩 해보도록 하겠습니다. 실시간으로 조건식에 편입되는 종목을 알려주는 이 기능을 확장하면 실시간으로 조건식에 편입되는 종목들을 바로 매수하는 기능을 구현 할 수 있습니다. 먼저 화면 구성입니다.

실시간 종목 검색 절차는 아래와 같습니다.

- 화면에 조건식 목록 바인딩 □
- conditionDataGridView에서 조건식 선택하면 화면에 종목 목록 바인딩□
- 실시간 편입 이탈 종목 요청 , 화면에 바인딩□

순서대로 코드를 작성해보도록 하겠습니다. 먼저 Form1.cs 의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 conditionDataGridView.SelectionChanged += DataGridView_SelectionChanged;

 axKHOpenAPI1.OnEventConnect += AxKHOpenAPI_OnEventConnect;
 axKHOpenAPI1.OnReceiveConditionVer += AxKHOpenAPI_OnReceiveConditionVer;
 axKHOpenAPI1.OnReceiveTrCondition += AxKHOpenAPI1_OnReceiveTrCondition;
 axKHOpenAPI1.OnReceiveRealCondition += AxKHOpenAPI1_OnReceiveRealCondition;
 axKHOpenAPI1.CommConnect();
}
```

화면에 조건식 목록을 바인딩하기 위해서는 먼저 OnEventConnect 함수에서 조건식 정보를 요청해야 합니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 axKHOpenAPI1.GetConditionLoad();
 }
}
```

GetConditionLoad 함수를 호출해서 사용자가 가지고 있는 조건식 정보를 요청합니다. GetConditionLoad 함수는 OnReceiveConditionVer 함수를 호출합니다. OnReceiveConditionVer 함수에서 사용자 조건식을 요청하고 화면에 바인딩 합니다.

```
public void onReceiveConditionVer(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 conditionObjectList = new List<ConditionObject>();
 if (e.lRet == 1)
 {
 string conditionList = axKHOpenAPI1.GetConditionNameList().TrimEnd(';');
 string[] conditionArray = conditionList.Split(';');

 for (int i = 0; i < conditionArray.Length; i++)
 {
 string[] condition = conditionArray[i].Split('^');
 conditionDataGridView.Rows.Add();
 conditionDataGridView["조건식_번호", i].Value = condition[0];
 conditionDataGridView["조건식_조건명", i].Value = condition[1];

 conditionObjectList.Add(new ConditionObject(condition[0],
condition[1]));
 }
 }
}
```

conditionDataGridView에 조건식 목록을 바인딩 합니다. DataGridView에 바인딩 되는 항목들은 모두 class를 생성해서 리스트로 관리해줍니다.

```
class ConditionObject
{
 public string conditionIndex;
 public string conditionName;
 public List<StockItem> stockItemList;
 public ConditionObject(string conditionIndex, string conditionName)
 {
 this.conditionIndex = conditionIndex;
 this.conditionName = conditionName;
 stockItemList = new List<StockItem>();
 }
}
```

조건식을 모델링한 클래스 ConditionObject에는 조건식으로 검색될 종목 리스트인 stockItemList 도 작성합니다. StockItem클래스는 아래와 같이 작성합니다. StockItem클래스는 종목코드 , 종목이름 , 현재가 , 등락율 , 전일대비 , 거래량을 필드로 갖습니다. 변수로 각각 stockCode , stockName , currentPrice , upDownRate , netChange, volume으로 정의합니다.

```
class StockItem
{
 public string stockCode;
 public string stockName;
 public long currentPrice;
 public double upDownRate;
 public int netChange;
 public long volume;
 public StockItem(string stockCode, string stockName, long currentPrice,
double upDownRate, int netChange, long volume)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 this.currentPrice = currentPrice;
 this.upDownRate = upDownRate;
 this.netChange = netChange;
 this.volume = volume;
 }
}
```

여기까지 코드를 작성했다면 조건식을 화면에 바인딩하는 과정이 끝났습니다.

- 화면에 조건식 목록 바인딩
- conditionDataGridView에서 조건식 선택하면 화면에 종목 목록 바인딩
- 실시간 편입 이탈 종목 요청 , 화면에 바인딩

다음은 conditionDataGridView에서 조건식을 선택하면 화면에 종목 정보들이 바인딩 되도록 코드를 작성하겠습니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
```

```

conditionDataGridView.SelectionChanged += conditionSearch;
axKHOpenAPI1.OnReceiveTrCondition += onReceiveTrCondition;
axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
axKHOpenAPI1.OnReceiveRealCondition += onReceiveRealCondition;
}

```

Form1.cs의 생성자에 추가된 함수 중에서 conditionSearch의 내용을 작성합니다. 조건식이 선택되었을 때 해당 조건식을 사용해서 종목 정보를 요청하면 됩니다. 호출할 함수는 SendCondition 입니다. 9.2장에서 작성한 내용과 같습니다.

```

public void conditionSearch(object sender, EventArgs e)
{
 try
 {
 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string conditionName = conditionDataGridView["조건식_조건명",
rowIndex].Value.ToString();
 int conditionIndex = int.Parse(conditionDataGridView["조건식_번호",
rowIndex].Value.ToString());
 axKHOpenAPI1.SendCondition("0156", conditionName, conditionIndex,
0);
 }
 }

 catch (Exception exception)
 {
 Console.WriteLine(exception.Message.ToString());
 }
}

```

화면 컨트롤에서 조건식 이름, 조건식 인덱스 정보를 가져오고 함수를 호출합니다.

SendCondition 함수는 OnReceiveTrCondition 함수를 호출합니다. OnReceiveTrCondition 함수에서는 조건식을 사용해서 검색된 종목코드들을 확인 할 수 있습니다. 얻어진 종목코드 리스트는 CommKwRqData 함수를 호출하는데 쓰일 것입니다.

[OnReceiveTrCondition() 이벤트]

```

OnReceiveTrCondition(
BSTR sScrNo, // 화면번호

```

```
BSTR strCodeList, // 종목코드 리스트
BSTR strConditionName, // 조건식 이름
int nIndex, // 조건명 인덱스
int nNext // 연속조회 여부
)
```

조건검색 요청으로 검색된 종목코드 리스트를 전달하는 이벤트 함수입니다.  
종목코드 리스트는 각 종목코드가 ';'로 구분되어 전달됩니다.

---



---

[조건검색 결과 수신예시]

```
OnReceiveTrCondition(LPCTSTR sScrNo, LPCTSTR strCodeList, LPCTSTR
strConditionName, int nIndex, int nNext)
{
 if(strCodeList == "") return;
 CString strCode, strCodeName;
 int nIdx = 0;
 while(AfxExtractSubString(strCode, strCodeList, nIdx++, T(';')))// 하나씩 종목코
드를 분리
 {
 if(strCode == T("")) continue;
 strCodeName = OpenAPI.GetMasterCodeName(strCode); // 종목명을 가져온다.
 }
}
```

CommKwRqData 함수를 호출하면 한번에 100종목의 조회요청을 할 수 있습니다.

[CommKwRqData() 함수]

```
CommKwRqData(
BSTR sArrCode, // 조회하려는 종목코드 리스트
BOOL bNext, // 연속조회 여부 0:기본값, 1:연속조회
int nCodeCount, // 종목코드 갯수
int nTypeFlag, // 0:주식 관심종목, 3:선물옵션 관심종목
BSTR sRQName, // 사용자 구분명
BSTR sScreenNo // 화면번호
)
```

한번에 100종목을 조회할 수 있는 관심종목 조회함수인데 영웅문HTS [0130] 관심종목 화면과는 이름만 같은 뿐 전혀 관련이 없습니다.

함수인자로 사용하는 종목코드 리스트는 조회하려는 종목코드 사이에 구분자 ';'를 추가해서 만들면 됩니다.

9.3장에서는 CommKwRqData 함수에 종목코드 리스트를 적어서 함수를 호출하도록 하겠습니다.

OnReceiveTrCondition 함수의 내용을 아래와 같이 작성합니다.

```
public void onReceiveTrCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrConditionEvent e)
{
 if (e.strCodeList.Length > 0)
 {
 string stockCodeList = e.strCodeList.Remove(e.strCodeList.Length - 1);
 int stockCount = stockCodeList.Split(';').Length;

 if (stockCount <= 100)
 {
 axKHOpenAPI1.CommKwRqData(stockCodeList, 0, stockCount, 0, "조건
검색종목", "5100");
 }
 if (e.nNext!=0)//연속조회여부 , 100개 이상 더 종목이 있을때 한번 더 조건검색
을 요청한다.
 {
 axKHOpenAPI1.SendCondition("5101", e.strConditionName, e.nIndex,
1);//
 }
 }
 else if (e.strCodeList.Length == 0)
 {
 MessageBox.Show("검색된 종목이 없습니다.");
 }
}
```

종목 코드 리스트가 있는지 먼저 확인합니다. 검색되는 종목이 없다면 검색된 종목이 없다는 메세지를 담은 MessageBox를 화면에 띄웁니다. 100종목을 조회요청 할 수 있는 함수이므로 if 문을 사용해서 종목코드의 개수가 100개 이하일 때 CommKwRqData 함수를 호출하도록 합니다.

CommKwRqData 함수를 호출하면 OnReceiveTrData 함수가 호출됩니다. OnReceiveTrData

함수에서 종목들의 정보를 요청하는 코드를 작성하겠습니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "조건검색종목")
 {

 }
}
```

OnReceiveTrData함수는 사용자가 보낼 수 있는 여러가지 요청에 대응하여 호출되는 함수입니다. 그래서 if문을 사용해서 이렇게 요청을 구분하는 습관을 들이시는 것이 좋습니다. if 문 안에 아래와 같이 코드를 작성합니다.

```
if (e.sRQName == "조건검색종목")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName); //요청의 반복 횟수를 요청합니다.

 int conditionObjectIndex = -1;

 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int conditionIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string selectedConditionName = conditionDataGridView["조건식_조건명",
conditionIndex].Value.ToString();
 conditionObjectIndex = conditionObjectList.FindIndex(o =>
o.conditionName == selectedConditionName);
 }

 conditionObjectList[conditionObjectIndex].stockItemList.Clear();
 stockDataGridView.Rows.Clear();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목명").Trim();
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Replace("-", ""));
 double upDownRate =
```

```

double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "등락율"));
 int netChange = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "전일대비"));
 long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "거래량"));

 stockDataGridView.Rows.Add();
 stockDataGridView["조건종목_종목코드", i].Value = stockCode;
 stockDataGridView["조건종목_종목명", i].Value = stockName;
 stockDataGridView["조건종목_현재가", i].Value = String.Format("{0:#,###}", currentPrice);
 stockDataGridView["조건종목_전일대비", i].Value = netChange;
 stockDataGridView["조건종목_등락율", i].Value = upDownRate;
 stockDataGridView["조건종목_거래량", i].Value = String.Format("{0:#,###}", volume);

 if (conditionObjectIndex > -1)
 {
 conditionObjectList[conditionObjectIndex].stockItemList.Add(new StockItem(stockCode, stockName, currentPrice, upDownRate, netChange, volume));
 }
}
}

```

현재 여러개의 종목코드들의 정보를 요청하고 있습니다. 여러 개의 종목 정보를 요청하려면 먼저 GetRepeatCnt 함수를 호출해서 몇번이나 종목 정보를 요청해야 하는지 요청 반복 횟수를 알아내야 합니다. 다음 conditionObjectIndex 를 -1로 초기화 했습니다. -1로 초기화하는 이유가 있습니다. 일단 이 코드에서는 지금 2가지 리스트가 있습니다.

- 조건식의 목록인 conditionObjectList
- 조건식으로 검색된 종목의 목록인 stockItemList

conditionObjectIndex가 만약에 -1이라면 조건식이 선택되지 않아서 아래 코드가 실행되지 않았다는 말이 됩니다.

```

if (conditionDataGridView.SelectedCells.Count > 0)
{
 int conditionIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string selectedConditionName = conditionDataGridView["조건식_조건명",
conditionIndex].Value.ToString();
 conditionObjectIndex = conditionObjectList.FindIndex(o => o.conditionName
== selectedConditionName);

```

```
{}
```

하지만 화면에서 선택된 조건식이 있어서 이 코드가 실행된다면 conditionIndex에는 0이상의 값이 저장되게 됩니다. conditionObjectList의 인덱스 값은 무조건 0부터 시작하기 때문입니다. 이렇게 0 이상의 값이 저장된다면 onReceiveTrData 함수의 하단에 있는 아래 코드가 동작하게 됩니다.

```
if (conditionObjectIndex > -1)
{
 conditionObjectList[conditionObjectIndex].stockItemList.Add(new
StockItem(stockCode, stockName, currentPrice, upDownRate, netChange,
volume));
}
```

이 코드는 conditionObjectList의 특정 인덱스에 먼저 접근합니다. 그렇게 하면 조건식(condition) 객체에 접근 할 수 있습니다. 하나의 Condition 객체는 stockItemList 를 가지고 있습니다. 조건식의 stockItemList에 검색된 종목들을 하나씩 추가합니다. 이 코드가 조건식이 선택 되었을때만 문제없이 실행되도록 conditionIndex의 초기값을 -1로 설정하는 것입니다.

코드의 중간부분을 보면 conditionObjectList[conditionObjectIndex]의 stockItemList 를 Clear 하고 있습니다. 사용자가 새롭게 조건식을 선택할때마다 새롭게 검색되는 종목들을 리스트에 담기 위함입니다. stockDataGridView도 종목 검색을 요청 할때마다 새로운 결과를 바인딩 하기 위해서 Clear 되고 있습니다.

```
conditionObjectList[conditionObjectIndex].stockItemList.Clear();
stockDataGridView.Rows.Clear();
for (int i = 0; i < count; i++)
{
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "종목명").Trim();
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가").Replace("-", ""));
 double upDownRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "등락율"));
 int netChange = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "전일대비"));
 long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "거래량"));
```

```

stockDataGridView.Rows.Add();
stockDataGridView["조건종목_종목코드", i].Value = stockCode;
stockDataGridView["조건종목_종목명", i].Value = stockName;
stockDataGridView["조건종목_현재가", i].Value = String.Format("{0:#,###}", currentPrice);
stockDataGridView["조건종목_전일대비", i].Value = netChange;
stockDataGridView["조건종목_등락율", i].Value = upDownRate;
stockDataGridView["조건종목_거래량", i].Value = String.Format("{0:#,###}", volume);

```

for문의 내부에서 종목 정보를 요청하고 화면에 바인딩 합니다. 여기까지 코드를 작성하면 조건식을 사용해서 검색되는 종목의 정보를 화면에 바인딩 할 수 있습니다.

- 화면에 조건식 목록 바인딩
- conditionDataGridView에서 조건식 선택하면 화면에 종목 목록 바인딩
- 실시간 편입 이탈 종목 요청, 화면에 바인딩

이제 실시간 편입, 이탈 종목 정보를 요청하고 화면에 바인딩 하도록 하겠습니다. 실시간 편입 이탈 정보는 OnReceiveRealCondition 함수에 담겨 있습니다. OnReceiveRealCondition 함수는 SendCondition 함수를 호출하면 호출됩니다. 그래서 별도의 호출 함수를 작성하지는 않고 OnReceiveRealCondition 함수의 내용만 작성합니다.

아래와 같은 순서로 코드가 동작하게 됩니다.

- SendCondition 으로 실시간 종목검색 요청
- OnReceiveRealCondition 이벤트 함수 호출
- OnReceiveRealCondition 이벤트 함수에서 종목 편입, 이탈 정보 확인
- 편입, 이탈 종목 정보 화면에 바인딩

KOA Studio에서는 아래와 같이 함수의 설명을 서술하고 있습니다.

[OnReceiveRealCondition() 이벤트]

```

OnReceiveRealCondition(
BSTR strCode, // 종목코드
BSTR strType, // 이벤트 종류, "I":종목편입, "D", 종목이탈
BSTR strConditionName, // 조건식 이름
BSTR strConditionIndex // 조건명 인덱스
)

```

실시간 조건검색 요청으로 신규종목이 편입되거나 기존 종목이 이탈될때 마다 호출됩니다.

-----  
-----

[실시간 조건검색 수신예시]

```
OnReceiveRealCondition(LPCTSTR sCode, LPCTSTR sType, LPCTSTR
strConditionName, LPCTSTR strConditionIndex)
{
 CString strCode(sCode), strCodeName;
 int nIdx = 0;
 CString strType(sType);
 if(strType == T("I"))// 종목편입
 {
 strCodeName = OpenAPI.GetMasterCodeName(strCode); // 종목명을 가져온다.
 long lRet = OpenAPI.SetRealReg(strSavedScreenNo, strCode,
T("9001;302;10;11;25;12;13"), "1");// 실시간 시세등록
 }
 else if(strType == T("D")) // 종목이탈
 {
 OpenAPI.SetRealRemove(strSavedScreenNo, strCode);// 실시간 시세해지
 }
}
```

---

KOA Studio의 예제처럼 아래와 같이 함수를 작성합니다. OnReceiveRealCondition의 이벤트 객체 e에서는 종목코드 , 이벤트 종류 , 조건식이름 , 조건명인덱스 4가지 정보를 확인할 수 있습니다.

```
public void onReceiveRealCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealConditionEvent e)
{
 if (e.strType == "I")//종목 편입
 {

 }
 else if (e.strType == "D")//종목 이탈
 {

 }
}
```

이제 종목이 편입되었을 때는 insertListBox에 종목 정보를 바인딩 하겠습니다. 종목이 이탈 되었을 때는 deleteListBox에 종목 정보를 바인딩 하도록 하겠습니다.

```
if (e.strType == "I")//종목 편입
{
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 insertListBox.Items.Add("종목편입| 조건인덱스 : "+e.strConditionIndex+" | 종목코드 : "+e.sTrCode+" | "+"종목명 : "+stockName);
}
```

종목이 편입 되었을 때 insertListBox에 문자열을 추가합니다.

```
else if (e.strType == "D")//종목 이탈
{
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 deleteListBox.Items.Add("종목이탈| 조건인덱스 : " + e.strConditionIndex + " | 종목코드 : " + e.sTrCode + " | " + "종목명 : " + stockName);
}
```

종목이 이탈 되었을 때의 코드도 작성합니다.

### 전체코드

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormsApp10
{
 public partial class Form1 : Form
 {
 List<ConditionObject> conditionObjectList;
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;

 axKHOpenAPI1.OnReceiveConditionVer += onReceiveConditionVer;
 conditionDataGridView.SelectionChanged += conditionSearch;
 axKHOpenAPI1.OnReceiveTrCondition += onReceiveTrCondition;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 }
 }
}
```

```

 axKHOpenAPI1.OnReceiveRealCondition += onReceiveRealCondition;
 }

 public void onReceiveRealCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealConditionEvent e)
{
 if (e.strType == "I")//종목 편입
 {
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 insertListBox.Items.Add("종목편입| 조건인덱스 : " +
" + e.strConditionIndex + " | 종목코드 : " + e.sTrCode + " | +" + "종목명 : " + stockName);
 }
 else if (e.strType == "D")//종목 이탈
 {
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 deleteListBox.Items.Add("종목이탈| 조건인덱스 : " +
e.strConditionIndex + " | 종목코드 : " + e.sTrCode + " | +" + "종목명 : " + stockName);
 }
}

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "조건검색종목")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);

 int conditionObjectIndex = -1;

 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int conditionIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string selectedConditionName = conditionDataGridView["조건식_조건명", conditionIndex].Value.ToString();
 conditionObjectIndex = conditionObjectList.FindIndex(o =>
o.conditionName == selectedConditionName);
 }
 }
}

```

```

 conditionObjectList[conditionObjectIndex].stockItemList.Clear();
 stockDataGridView.Rows.Clear();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가").Replace("-", ""));
 double upDownRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "등락율"));
 int netChange = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "전일대비"));
 long volume = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "거래량"));

 stockDataGridView.Rows.Add();
 stockDataGridView["조건종목_종목코드", i].Value = stockCode;
 stockDataGridView["조건종목_종목명", i].Value = stockName;
 stockDataGridView["조건종목_현재가", i].Value = String.Format("{0:#,###}", currentPrice);
 stockDataGridView["조건종목_전일대비", i].Value = netChange;
 stockDataGridView["조건종목_등락율", i].Value = upDownRate;
 stockDataGridView["조건종목_거래량", i].Value = String.Format("{0:#,###}", volume);

 if (conditionObjectIndex > -1)
 {

conditionObjectList[conditionObjectIndex].stockItemList.Add(new StockItem(stockCode, stockName, currentPrice, upDownRate, netChange, volume));
 }
 }
 }

 public void onReceiveTrCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrConditionEvent e)
{
}

```

```

 if (e.strCodeList.Length > 0)
 {
 string stockCodeList = e.strCodeList.Remove(e.strCodeList.Length
- 1);
 int stockCount = stockCodeList.Split(';').Length;

 if (stockCount <= 100)
 {
 axKHOpenAPI1.CommKwRqData(stockCodeList, 0, stockCount,
0, "조건검색종목", "5100");
 }
 if (e.nNext!=0)
 {
 axKHOpenAPI1.SendCondition("5101", e.strConditionName,
e nIndex, 1); //실시간 조건검색 , OnReceiveRealConditoin 함수 호출
 }
 else if (e.strCodeList.Length == 0)
 {
 MessageBox.Show("검색된 종목이 없습니다.");
 }
 }
 public void conditionSearch(object sender, EventArgs e)
 {
 try
 {
 if (conditionDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = conditionDataGridView.SelectedCells[0].RowIndex;
 string conditionName = conditionDataGridView["조건식_조건명",
rowIndex].Value.ToString();
 int conditionIndex = int.Parse(conditionDataGridView["조건식_
번호", rowIndex].Value.ToString());
 axKHOpenAPI1.SendCondition("0156", conditionName,
conditionIndex, 0);
 }
 }
 catch (Exception exception)
 {

```

```

 Console.WriteLine(exception.Message.ToString());
 }

}

public void onReceiveConditionVer(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveConditionVerEvent e)
{
 conditionObjectList = new List<ConditionObject>();
 if (e.lRet == 1)
 {
 string conditionList =
axKHOpenAPI1.GetConditionNameList().TrimEnd('\'');
 string[] conditionArray = conditionList.Split('\'');

 for (int i = 0; i < conditionArray.Length; i++)
 {
 string[] condition = conditionArray[i].Split('^');
 conditionDataGridView.Rows.Add();
 conditionDataGridView["조건식_번호", i].Value = condition[0];
 conditionDataGridView["조건식_조건명", i].Value = condition[1];

 conditionObjectList.Add(new ConditionObject(condition[0],
condition[1]));
 }
 }
}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode==0)
 {
 axKHOpenAPI1.GetConditionLoad();
 }
}

class ConditionObject
{
 public string conditionIndex;
 public string conditionName;
 public List<StockItem> stockItemList;
 public ConditionObject(string conditionIndex, string conditionName)
 {

```

```
 this.conditionIndex = conditionIndex;
 this.conditionName = conditionName;
 stockItemList = new List<StockItem>();
 }
}

class StockItem
{
 public string stockCode;
 public string stockName;
 public long currentPrice;
 public double upDownRate;
 public int netChange;
 public long volume;
 public StockItem(string stockCode, string stockName, long currentPrice,
double upDownRate, int netChange, long volume)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 this.currentPrice = currentPrice;
 this.upDownRate = upDownRate;
 this.netChange = netChange;
 this.volume = volume;
 }
}
```

실행화면

## 10. 조건식 기반 자동매매

### Preview

10장에서는 1~9장에서 배운 내용을 바탕으로 자동매매 프로그램을 만들어봅니다. 앞에서 학습한 내용들을 하나씩 되짚어보면서 프로그램을 만들어봅니다.

### Contents

10.1장에서는 프로그램의 화면을 구성합니다.

10.2장에서는 종목 매수주문 기능을 구현합니다. 6장에서 학습했던 내용과 같습니다.

10.3장에서는 잔고를 조회합니다. 잔고를 조회했던 7.1장의 내용을 떠올려봅니다.

10.4장에서는 주문기록을 화면에 출력합니다. 8장에서 배웠던 OnReceiveChejanData 함수를 사용합니다.

10.5장에서는 종목 매도주문 기능을 구현합니다. 7.2장에서 배웠던 SendOrder 함수를 사용해봅니다.

10.6장에서는 자동매매 거래 규칙을 설정합니다.

10.7장에서는 주문을 정정하고 취소합니다. 먼저 8.1장에서 학습한 내용을 바탕으로 미체결 주문을 출력합니다. 그리고 SendOrder 함수를 사용해서 주문을 정정하고 취소합니다.

10.8장에서는 자동매매 프로그램을 실행해서 종목들을 매수해봅니다.

10.9장에서는 실행중인 자동매매 거래규칙을 중지해봅니다. 사용자가 가지고 있는 주식을 일괄 매도하는 방법도 알아봅니다.

## 10.1 프로그램 화면 구성 / 설명

10.1장에서는 조건식 기반 자동매매를 하기전에 당일 매수 금액 제한을 설정하는 방법에 대해서 알아보도록 하겠습니다. 먼저 화면을 구성합니다. 지금 구성하는 화면은 10.1, 10.2, 10.3장에서 계속 사용하게 될 화면입니다. 10장은 이전 내용들보다 소스코드의 길이가 길어질 것입니다. 그래서 10.1장에 모든 화면 구성에 관한 내용을 담고 10.2, 10.3, 10.4장에서는 화면 구성은 생략하도록 하겠습니다.

### 화면구성

10장에서 만드는 프로그램의 동작 설명

#### 매수규칙 사용 방법

- conditionComboBox에서 매수금액 제한을 설정할 조건식 선택 한다.
- 해당 조건식으로 매입 할 수 있는 전체 금액을 설정한다.
- 해당 조건식으로 매수 할 수 있는 전체 종목 개수를 설정한다.
- 종목당 매입금액 Label에 매입제한 금액 / 매입제한 종목 개수 값이 바인딩 된다.
- 매수 거래구분 ComboBox에서는 시장가, 지정가를 선택 할 수 있다.
- 시장가를 선택하면 시장가 매수한다. 지정가를 선택하면 현재 가로 매수한다.
- 매수규칙 설정을 클릭하면 조건식의 매수 규칙이 설정된다.
- 자동매매 시작 버튼을 클릭하면 설정한 조건식의 실시간 편입 종목을 매수한다.

#### 매도규칙 사용 방법

- 이익률 NumericUpDown에서는 보유한 주식이 몇 % 오르면 매도 주문을 요청할지 설정한다.
- 손절률 NumericUpDown에서는 보유한 주식이 몇 % 하락하면 매도 주문을 요청할지 설정한다.
- 매도 거래구분 ComboBox에서는 시장가 , 지정가를 선택 할

수 있다.

- 시장가를 선택하면 시장가 매도한다. 지정가를 선택하면 현재 가로 매도한다.
- 전체 청산 버튼을 클릭하면 보유하고 있는 모든 주식을 시장 가 매도한다.

잔고 조회, 정정, 주문취소 사용 방법

잔고조회 버튼을 클릭해서 요청 할 수 있는 정보는 다음과 같다.

- 사용자 주식 잔고 / 미체결
- 예수금 / 총매입금액 / 총평가금액
- 당일손익금액 / 당일손익율 / 실현손익
- 정정 버튼 사용 방법
- 미체결 DataGridView에서 주문을 선택하고 가격, 수량을 입력 한 후 정정 버튼을 클릭하면 주문을 정정 할 수 있다.
- 주문취소 버튼 사용 방법
- 미체결 DataGridView에서 주문을 선택하고 주문 취소 버튼을 클릭하면 주문을 취소한다.

매수주문, 매도주문 사용 방법

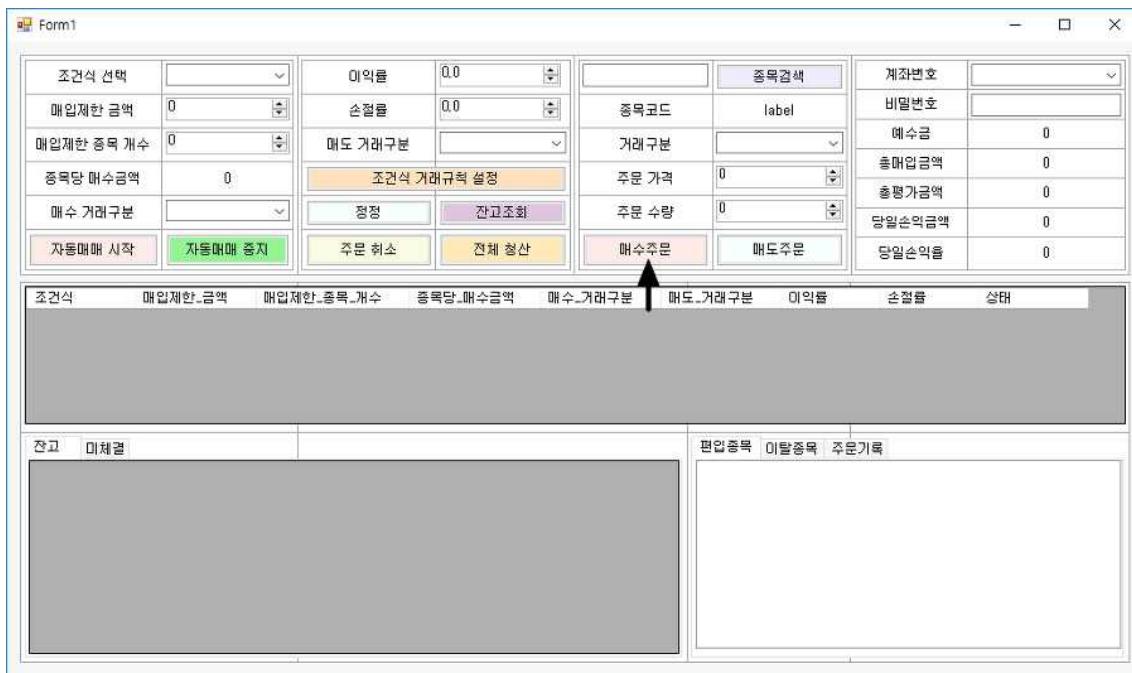
- 종목 TextBox에 종목이름을 입력하고 종목 정보를 검색 할 수 있다.
- 종목을 검색하면 종목코드, 현재가격이 컨트롤에 바인딩 된다.
- 거래구분 ComboBox의 값을 지정한다.
- 가격 NumericUpDown, 수량 NumericUpDown 값을 설정한다.
- 매수주문 버튼을 클릭하면 종목의 매수 주문이 요청된다.
- 잔고 DataGridView에서 주식 잔고를 선택하고 매도주문 버튼을 클릭하면 매도 주문이 요청된다.

10장에서 만들게 될 프로그램의 기능은 크게 3가지입니다.

- 실시간으로 사용자 조건식에 편입되는 종목 매수주문
- 조건식마다 사용자가 설정한 금액 만큼만 매수
- 매수한 종목은 사용자가 설정한 이익 규칙에 따라서 매도

## 10.2 종목 매수 주문

10.2 장에서는 화면의 매수주문을 클릭했을 때 동작할 코드를 작성하도록 하겠습니다. 이번 10.2장의 내용은 6.1장의 내용과 유사합니다.



종목을 주문하기 위해서는 계좌번호, 종목코드, 거래구분, 주문가격, 주문수량 정보가 필요합니다. 종목코드 정보는 stockTextBox에 종목이름을 적고 종목검색 Button을 클릭했을 때 종목코드 Label에 바인딩 되도록 합니다.

종목검색 버튼을 클릭했을 때 동작할 함수를 Form1.cs의 생성자에 추가합니다.

```
public Form1()
{
 InitializeComponent();

 stockSearchButton.Click += ButtonClicked;
 axKHOpenAPI1.CommConnect();

}
```

함수의 동작을 정의합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{

 else if (sender.Equals(stockSearchButton))
 {
 string stockName = stockTextBox.Text;
 int index = stockList.FindIndex(o => o.stockName == stockName);
 string stockCode = stockList[index].stockCode;
 stockCodeLabel.Text = stockCode;

 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.CommRqData("종목정보요청", "opt10001", 0, "5000");
 }

}
```

stockSearchButton을 클릭했을때 stockList에서 종목명과 일치하는 종목코드를 찾아서 화면에 바인딩합니다. 종목의 현재 가격을 알아내기 위해서 찾아낸 종목코드를 사용해서 CommRqData 함수를 호출합니다. CommRqData 함수는 OnReceiveTrData 이벤트 함수를 호출하게 됩니다. 매수 주문 전에 OnReceiveTrData 함수에서 종목의 현재 가격을 요청하고 화면에 바인딩합니다.

Form1.cs의 생성자에 코드를 추가합니다

```
public Form1()
{
 InitializeComponent();

 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 axKHOpenAPI1.CommConnect();

}
```

함수를 추가했다면 함수의 내용을 작성합니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{

 if (e.sRQName == "종목정보요청")
 {
 string currentStockPrice = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "현재가");
 orderPriceNumericUpDown.Value =
long.Parse(currentStockPrice.Replace("-", ""));
 }

}
```

GetCommData 함수를 호출해서 종목의 현재가격을 요청하고 화면에 바인딩 합니다. 이 내용은 4.1장의 내용과 같습니다. 다음은 매수주문 버튼을 클릭했을때 동작할 코드를 작성합니다. Form1.cs의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 buyButton.Click += ButtonClicked;
 axKHOpenAPI1.CommConnect();

}
```

추가한 ButtonClicked함수의 내용을 작성합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {
 string buyOrderType = orderComboBox.Text;
 int buyOrderPrice =
int.Parse(orderPriceNumericUpDown.Value.ToString());
```

```

int buyOrderNumber =
int.Parse(orderNumberNumericUpDown.Value.ToString());
 string accountNumber = accountComboBox.Text;
 string stockCode = stockCodeLabel.Text;

 if (buyOrderType.Length>0 && buyOrderPrice>0 && buyOrderNumber>0
&& accountNumber.Length>0 && stockCode.Length>0)
 {
 string[] orderType = buyOrderType.Split(':');
 axKHOpenAPI.SendOrder("신규종목매수주문",
", "8249", accountNumber, 1, stockCode, buyOrderNumber, buyOrderPrice, orderType[0],
"");

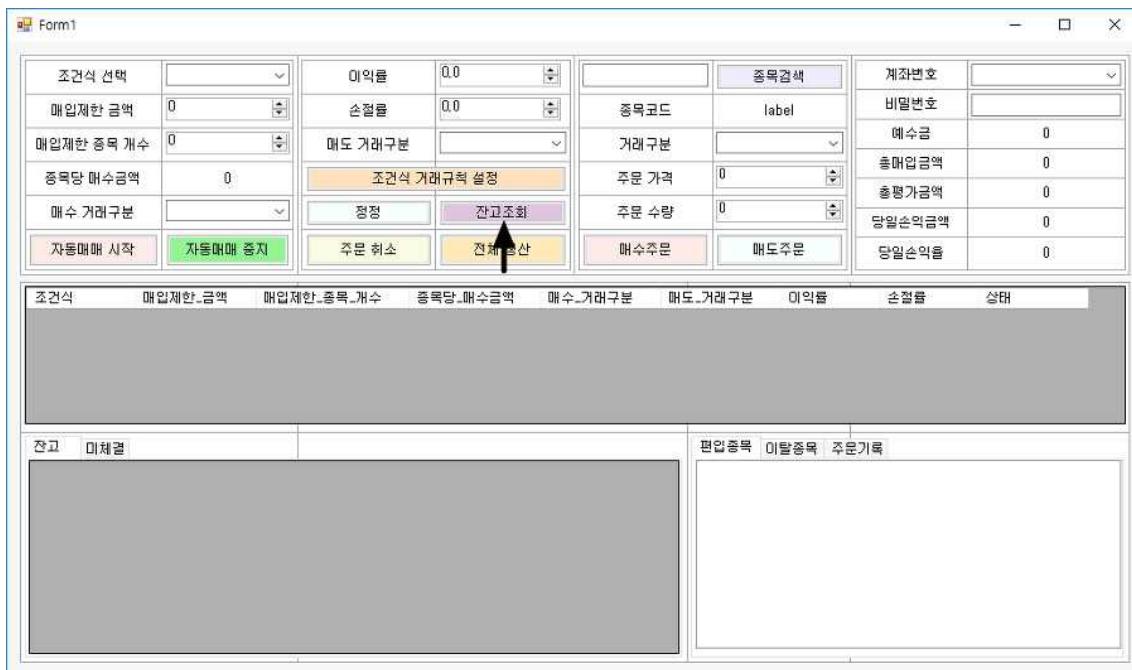
 }
}
}

```

매수주문 버튼을 클릭하면 화면 컨트롤에 값이 들어있는지 확인하고 SendOrder 함수를 호출합니다. 매수 주문에는 계좌번호, 종목코드, 주문가격, 주문수량, 거래구분 정보가 필요합니다. 함수의 호출에 필요한 인자값을 화면 컨트롤로부터 받아오고 SendOrder 함수를 호출하면 매수주문이 요청됩니다.

## 10.3 잔고 조회

10.3장에서는 화면의 잔고조회 버튼을 클릭했을 때 동작할 코드를 작성하도록 하겠습니다.  
이번 10.3장의 내용은 체결정보 8.1장, 3.3장의 내용과 유사합니다.



키움증권 API를 사용해서 잔고를 조회하려면 CommRqData -> 이벤트함수 -> 요청함수 순서대로 함수를 작성하고 호출하면 됩니다.

잔고 조회 버튼을 클릭했을 때 CommRqData 함수를 호출할 함수를 Form1.cs의 생성자에 추가합니다.

```
public Form1()
{
 InitializeComponent();

 balanceCheckButton.Click += ButtonClicked;
 axKHOpenAPI1.CommConnect();

}
```

잔고 조회 Button을 클릭했을때 사용자의 주식잔고 , 미체결주문 , 사용자의 계좌 잔액 정보를 요청하도록 하겠습니다. 10.2장에 이어서 10.3장의 내용을 학습하고 있다면 ButtonClicked 함수에는 10.2장의 코드가 작성되어 있을 것입니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(buyButton))
 {

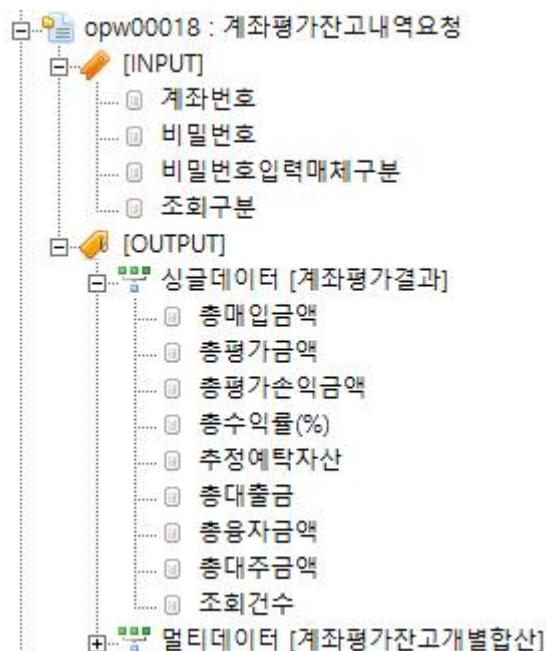
 }
 else if (sender.Equals(balanceCheckButton))
 {
 if (accountComboBox.Text.Length > 0 && passwordTextBox.Text.Length >
0)
 {
 string accountNumber = accountComboBox.Text;
 string password = passwordTextBox.Text;

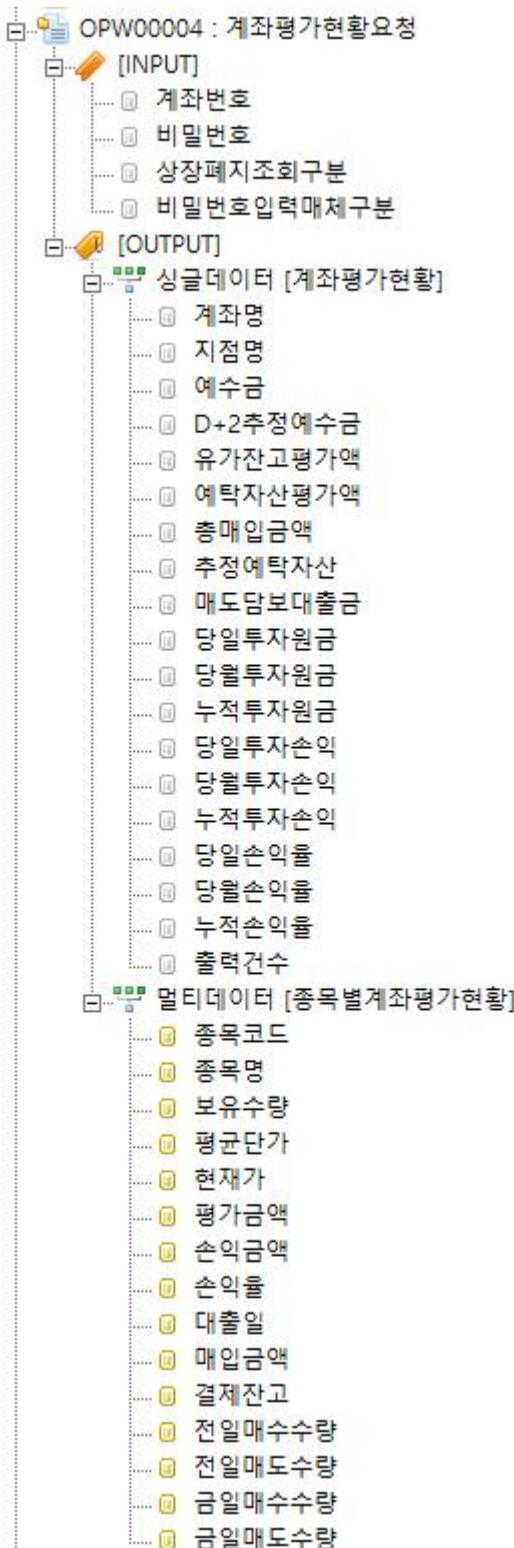
 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.SetInputValue("조회구분", "1");
 axKHOpenAPI1.CommRqData("계좌평가잔고내역요청", "opw00018", 0,
"8100");

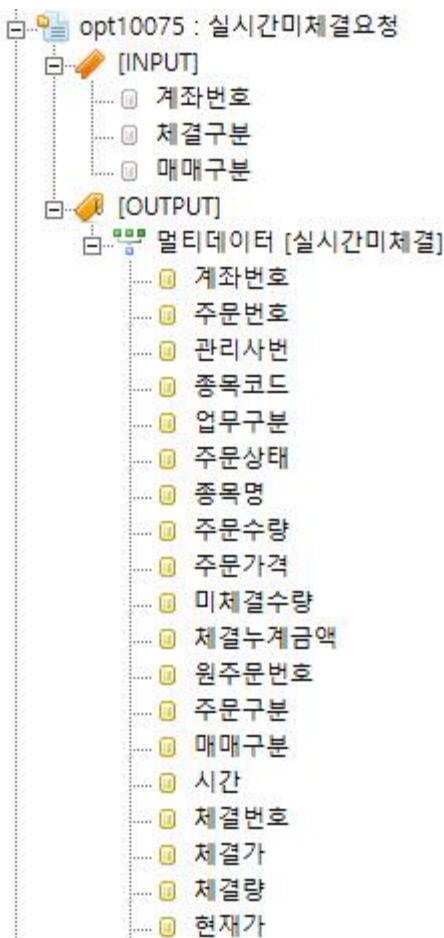
 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("비밀번호", password);
 axKHOpenAPI1.SetInputValue("상장폐지조회구분", "0");
 axKHOpenAPI1.SetInputValue("비밀번호입력매체구분", "00");
 axKHOpenAPI1.CommRqData("계좌평가현황요청", "opw00004", 0, "4000");

 axKHOpenAPI1.SetInputValue("계좌번호", accountNumber);
 axKHOpenAPI1.SetInputValue("체결구분", "1");
 axKHOpenAPI1.SetInputValue("매매구분", "2");
 axKHOpenAPI1.CommRqData("실시간미체결요청", "opt10075", 0, "5700");
 }
 }
}
```

else if 문을 사용해서 요청이 일어난 화면 컨트롤을 buyButton과 구분합니다. 아래 그림들은 CommRqData 함수를 호출해서 요청한 TR목록의 KOA Studio 화면입니다.








---

화면의 오른쪽에는 예수금, 총매입금액, 총평가금액, 당일손익금액, 당일손익율 Label이 있습니다. TR목록에 따르면 예수금, 계좌 주식 잔고는 "계좌평가현황요청" TR을 통해서 확인 할 수 있습니다. 총매입금액, 총평가금액은 "계좌평가잔고내역요청" TR을 통해서 확인 할 수 있습니다. 미체결 정보는 "실시간미체결요청" TR을 통해서 확인 할 수 있습니다. CommRqData 함수를 사용해서 요청을 모두 전송했다면 이벤트 함수를 정의합니다.

OnReceiveTrData 이벤트 함수에서 요청함수를 호출 할 수 있습니다.

Form1.cs의 생성자에 OnReceiveTrData 함수를 작성합니다.

```
public Form1()
{
 InitializeComponent();

 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;
 axKHOpenAPI1.CommConnect();
```

```
.....
.....
}
```

함수의 내용을 작성합니다. 앞서 보낸 3가지 요청을 처리하기 위한 코드를 onReceiveTrData 함수 안에서 if문으로 분리해서 작성합니다.

```
public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "계좌평가잔고내역요청")
 {

 }
 else if (e.sRQName== "계좌평가현황요청")
 {

 }
 else if (e.sRQName == "실시간미체결요청")
 {

 }

}
```

계좌평가 잔고내역 요청 if문의 안쪽에 총매입금액 , 총평가금액을 요청하는 함수를 작성합니다.

```
if (e.sRQName == "계좌평가잔고내역요청")
{
 long totalBuyingAmount = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "총매입금액"));
 long totalEstimatedAmount = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "총평가금액"));

 totalBuyLabel.Text = String.Format("{0:#,###}", totalBuyingAmount);
 totalEstimateLabel.Text = String.Format("{0:#,###}", totalEstimatedAmount);
}
```

GetCommData 함수를 호출해서 정보를 요청하고 화면에 바인딩합니다. 다음은 "계좌평가현황요청"을 작성합니다.

```
else if (e.sRQName== "계좌평가현황요청")
{
 long deposit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "예수금"));
 long todayProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "당일투자손익"));
 double todayProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "당일손익율
"));

 depositLabel.Text = String.Format("{0:#,###}", deposit);
 todayProfitLabel.Text = String.Format("{0:#,###}", todayProfit);
 todayProfitRateLabel.Text = String.Format("{0:#.##}", todayProfitRate);
}
```

예수금 , 당일투자손익 , 당일손익율을 먼저 요청합니다. 그리고 주식 잔고 정보를 요청하는 코드를 작성합니다.

```
else if (e.sRQName== "계좌평가현황요청")
{
 long deposit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "예수금"));
 long todayProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "당일투자손익"));
 double todayProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "당일손익율
"));

 depositLabel.Text = String.Format("{0:#,###}", deposit);
 todayProfitLabel.Text = String.Format("{0:#,###}", todayProfit);
 todayProfitRateLabel.Text = String.Format("{0:#.##}", todayProfitRate);

 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 stockBalanceList = new List<stockBalance>();
 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
```

```

i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "보유수량"));
 long buyingMoney = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "매입금액"));
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Replace("-", ""));
 long estimatedProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "손익금액"));
 double estimatedProfitRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalanceList.Add(new stockBalance(stockCode, stockName, number,
String.Format("{0:#,###}", buyingMoney), String.Format("{0:#,###}", currentPrice),
estimatedProfit, String.Format("{0:f2}", estimatedProfitRate)));
}
balanceDataGridView.DataSource = stockBalanceList;
}
}

```

GetRepeatCnt 함수를 호출해서 주식 잔고를 몇번 요청해야 하는지 알아냅니다.  
GetCommData 함수를 호출해서 요청 횟수만큼 종목 정보를 요청합니다. 다음은 미체결 주문 정보를 요청합니다.

```

else if (e.sRQName == "실시간미체결요청")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 outstandingList = new List<outstanding>();
 for (int i = 0; i < count; i++)
 {
 string orderCode = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문번호")).ToString();
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목명").Trim();
 int orderNumber = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문수량"));
 int orderPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문가격"));
 int outstandingNumber = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "미체결수량"));
 }
}

```

```

 int currentPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Replace("-", ""));
 string orderGubun = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문구분").Trim();
 string orderTime = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "시간").Trim();

 outstandingList.Add(new outstanding(orderCode, stockCode, stockName,
orderNumber, String.Format("{0:#,###}", orderPrice), String.Format("{0:#,###}",
currentPrice), outstandingNumber, orderGubun, orderTime));

 }
 outstandingDataGridView.DataSource = outstandingList;
}

```

실시간 미체결 주문 정보도 GetRepeatCnt 함수와 GetCommData 함수를 호출해서 요청합니다. 아래와 같이 코드를 작성 할 수 있습니다.

```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName == "계좌평가잔고내역요청")
 {
 long totalBuyingAmount =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "총매입금액"));
 long totalEstimatedAmount =
long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "총평가금액"));

 totalBuyLabel.Text = String.Format("{0:#,###}", totalBuyingAmount);
 totalEstimateLabel.Text = String.Format("{0:#,###}",
totalEstimatedAmount);
 }
 else if (e.sRQName == "계좌평가현황요청")
 {
 long deposit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "예수금"));
 long todayProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, 0, "당일투자손익"));
 double todayProfitRate =
double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "당일손익율
"));
 }
}

```

```

depositLabel.Text = String.Format("{0:#,###}", deposit);
todayProfitLabel.Text = String.Format("{0:#,###}", todayProfit);
todayProfitRateLabel.Text = String.Format("{0:#.##}", todayProfitRate);

int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
stockBalanceList = new List<stockBalance>();
for (int i = 0; i < count; i++)
{
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").TrimStart('0');
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 long number = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "보유수량"));
 long buyingMoney = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "매입금액"));
 long currentPrice = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가")
.Replace("-", ""));
 long estimatedProfit = long.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익금액"));
 double estimatedProfitRate = double.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "손익율"));

 stockBalanceList.Add(new stockBalance(stockCode, stockName,
number, String.Format("{0:#,###}", buyingMoney), String.Format("{0:#,###}", currentPrice),
estimatedProfit, String.Format("{0:f2}", estimatedProfitRate)));
}
balanceDataGridView.DataSource = stockBalanceList;
}

else if (e.sRQName == "실시간미체결요청")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);
 outstandingList = new List<outstanding>();
 for (int i = 0; i < count; i++)
 {
 string orderCode = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문번호")).ToString();
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").Trim();
 }
}

```

```
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 int orderNumber = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문수량"));
 int orderPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문가격"));
 int outstandingNumber =
int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "미체결수량"));
 int currentPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Replace("-", ""));
 string orderGubun = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "주문구분").Trim();
 string orderTime = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "시간").Trim();

 outstandingList.Add(new outstanding(orderCode, stockCode,
stockName, orderNumber, String.Format("{0:#,###}", orderPrice),
String.Format("{0:#,###}", currentPrice), outstandingNumber, orderGubun,
orderTime));

 }
 outstandingDataGridView.DataSource = outstandingList;
}
.....
.....
.....
}
```

## 실행 화면

Form 1

|                                                            |                      |             |                      |       |        |             |
|------------------------------------------------------------|----------------------|-------------|----------------------|-------|--------|-------------|
| 조건식 선택                                                     | <input type="text"/> | 미익률         | 0.0                  | 증목검색  | 계좌번호   | 8101317911  |
| 매입제한 금액                                                    | 0                    | 손절률         | 0.0                  | 증목코드  | 비밀번호   | ****        |
| 매입제한 증목 개수                                                 | 0                    | 매도 거래구분     | <input type="text"/> | 거래 구분 | 예수금    | 496,466,223 |
| 증목당 매수금액                                                   | 0                    | 조건식 거래규칙 설정 |                      | 주문 가격 | 총매입금액  | 4,780,480   |
| 매수 거래구분                                                    | <input type="text"/> | 정정          | 잔고조회                 | 주문 수량 | 총평가금액  | 4,767,365   |
| 자동매매 시작                                                    | 자동매매 중지              | 주문 취소       | 전체 청산                | 매수주문  | 당일손익금액 |             |
| 조건식 매입제한_금액 매입제한_증목_개수 증목당_매수금액 매수_거래구분 매도_거래구분 미익률 손절률 상태 |                      |             |                      |       |        |             |

↓

| 잔고 미체결  |         | 평입증목 이월증목 주문기록 |           |        |        |
|---------|---------|----------------|-----------|--------|--------|
| 증목코드    | 증목명     | 수량             | 매수금       | 현재가    | 평가순익   |
| A000547 | 중국화재2우B | 20             | 700,000   | 45,500 | 210000 |
| A000720 | 현대건설    | 20             | 1,480,000 | 69,300 | -94000 |
| A005030 | 부산주공    | 30             | 30,750    | 983    | -1260  |
| A005930 | 삼성전자    | 1              | 49,200    | 49,850 | 650    |
| A007390 | 네이처셀    | 10             | 305,500   | 29,850 | -7000  |
| A008250 | 미건산업    | 20             | 289,000   | 13,300 | -23000 |

Form 1

|                                                            |                      |             |                      |       |        |             |
|------------------------------------------------------------|----------------------|-------------|----------------------|-------|--------|-------------|
| 조건식 선택                                                     | <input type="text"/> | 미익률         | 0.0                  | 증목검색  | 계좌번호   | 8101317911  |
| 매입제한 금액                                                    | 0                    | 손절률         | 0.0                  | 증목코드  | 비밀번호   | ****        |
| 매입제한 증목 개수                                                 | 0                    | 매도 거래구분     | <input type="text"/> | 거래 구분 | 예수금    | 496,466,223 |
| 증목당 매수금액                                                   | 0                    | 조건식 거래규칙 설정 |                      | 주문 가격 | 총매입금액  | 4,780,480   |
| 매수 거래구분                                                    | <input type="text"/> | 정정          | 잔고조회                 | 주문 수량 | 총평가금액  | 4,723,370   |
| 자동매매 시작                                                    | 자동매매 중지              | 주문 취소       | 전체 청산                | 매수주문  | 당일손익금액 |             |
| 조건식 매입제한_금액 매입제한_증목_개수 증목당_매수금액 매수_거래구분 매도_거래구분 미익률 손절률 상태 |                      |             |                      |       |        |             |

↓

| 잔고 미체결 |        | 평입증목 이월증목 주문기록 |      |         |       |
|--------|--------|----------------|------|---------|-------|
| 주문번호   | 증목코드   | 증목명            | 주문수량 | 주문가격    | 미체결수량 |
| 66147  | 034730 | SK             | 50   | 290,000 | 50    |
| 66015  | 079940 | 가비아            | 30   | 7,000   | 30    |
| 65902  | 005930 | 삼성전자           | 30   | 40,000  | 30    |

## 10.4 주문 기록

10.4장에서는 사용자가 주문을 요청한 기록을 바인딩하도록 하겠습니다. 8.1장의 OnReceiveChejanData 함수를 활용하면 됩니다. OnReceiveChejanData는 사용자의 주식 주문, 체결 정보를 담고 있는 이벤트 함수입니다.

|            |         |            |          |          |          |     |     |       |        |
|------------|---------|------------|----------|----------|----------|-----|-----|-------|--------|
| 조건식 선택     | 매입제한 금액 | 매입제한 종목 개수 | 증목당 매수금액 | 매수 거래구분  | 매도 거래구분  | 미익률 | 손절률 | 증목검색  | 계좌번호   |
| 매입제한 금액    | 0       | 0          | 0        | 증목당 매수금액 | 증목당 매수금액 | 0.0 | 0.0 | label | 비밀번호   |
| 매입제한 종목 개수 | 0       | 0          | 0        | 매수 거래구분  | 매수 거래구분  |     |     | 거래 구분 | 예수금    |
| 증목당 매수금액   | 0       | 0          | 0        | 매도 거래구분  | 매도 거래구분  |     |     | 주문 가격 | 총매입금액  |
| 매수 거래구분    | 자동매매 시작 | 자동매매 종지    | 자동매매 종지  | 증목당 매수금액 | 증목당 매수금액 | 0.0 | 0.0 | 주문 수량 | 총평가금액  |
| 자동매매 시작    | 자동매매 종지 | 자동매매 종지    | 자동매매 종지  | 자동매매 종지  | 자동매매 종지  |     |     | 매수주문  | 당일손익금액 |
|            |         |            |          |          |          |     |     | 매도주문  | 당일손익률  |

|     |         |            |          |         |         |     |     |    |
|-----|---------|------------|----------|---------|---------|-----|-----|----|
| 조건식 | 매입제한_금액 | 매입제한_종목_개수 | 증목당_매수금액 | 매수_거래구분 | 매도_거래구분 | 미익률 | 손절률 | 상태 |
| 조건식 | 매입제한_금액 | 매입제한_종목_개수 | 증목당_매수금액 | 매수_거래구분 | 매도_거래구분 | 미익률 | 손절률 | 상태 |

|    |     |      |      |      |
|----|-----|------|------|------|
| 장고 | 미체결 | 평입증목 | 미탈증목 | 주문기록 |
| 장고 | 미체결 | 평입증목 | 미탈증목 | 주문기록 |

Form1.cs 의 생성자에 OnReceiveChejanData 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 axKHOpenAPI1.OnReceiveChejanData += onReceiveChejanData;
 axKHOpenAPI1.CommConnect();

}
```

함수를 생성자에 추가했으면 함수의 내용을 작성합니다. OnReceiveChejanData 함수는 e.sGubun 의 값으로 아래의 2가지 상태를 구분합니다.

- 주문 접수 / 체결
- 주식 잔고

2가지 상태를 if문을 사용해서 구분하고 코드를 작성합니다. 8.1장에서 했던 것처럼 GetChejanData() 함수를 호출해서 정보를 요청합니다.

```
public void onReceiveChejanData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveChejanDataEvent e)
{
 if (e.sGubun == "0")//주문 접수 , 체결시
 {
 string orderNumber = axKHOpenAPI1.GetChejanData(9203);
 string orderStatus = axKHOpenAPI1.GetChejanData(913);
 string orderStockName = axKHOpenAPI1.GetChejanData(302);
 string orderStockNumber = axKHOpenAPI1.GetChejanData(900);
 long orderPrice = long.Parse(axKHOpenAPI1.GetChejanData(901));
 string orderType = axKHOpenAPI1.GetChejanData(905);

 orderRecordListBox.Items.Add("주문번호 : "+orderNumber+ " | "+"주문상태 :
"+orderStatus);
 orderRecordListBox.Items.Add("종목명 : " + orderStockName + " | " + "주문
수량 : " + orderStockNumber);
 orderRecordListBox.Items.Add("주문가격 : " + String.Format("{0:#,###}",orderPrice));
 }
}
```

```

orderRecordListBox.Items.Add("주문구분 : " + orderType);

orderRecordListBox.Items.Add("-----");
}

else if (e.sGubun == "1")//국내주식 잔고전달
{
 string stockName = axKHOpenAPI1.GetChejanData(302);
 long currentPrice = long.Parse(axKHOpenAPI1.GetChejanData(10).Replace("-", ""));
}

string profitRate = axKHOpenAPI1.GetChejanData(8019);
long totalBuyingPrice = long.Parse(axKHOpenAPI1.GetChejanData(932));
long profitMoney = long.Parse(axKHOpenAPI1.GetChejanData(950));

todayProfitLabel.Text = String.Format("{0:#,###}", profitMoney);
todayProfitRateLabel.Text = profitRate;

}
}

```

주문 접수 , 체결시 e.sGubun의 값은 "0" 이 됩니다. e.sGubun의 값이 "0"이면 변수에 주문 정보를 저장하고 orderRecordListBox 에 바인딩합니다. 국내주식 잔고 전달시 e.sGubun의 값은 "1"이 됩니다. e.sGubun의 값이 "1" 이면 주식 잔고 정보를 요청하고 화면에 바인딩 합니다.

## 10.5 종목 매도 주문

10.5장에서는 사용자의 주식 잔고에서 종목을 선택해서 매도주문을 요청해보도록 하겠습니다. 7.2장의 주식 매도 주문하기를 참고하면 됩니다.

The screenshot shows a Windows application window titled "Form1". The interface is organized into several sections:

- Top Left:** Input fields for "조건식 선택" (Condition Selection), "매입제한 금액" (Purchase Limit Amount), and "매입제한 종목 개수" (Number of Stocks for Purchase Limit).
- Top Right:** Details section with dropdowns for "증목검색" (Stock Search), "증목코드" (Stock Code), "거래 구분" (Trade Type), and "증정" (Delivery). It also displays "계좌번호" (Account Number), "비밀번호" (Password), "예수금" (Balancing Fund), "총매입금액" (Total Purchase Amount), "총평가금액" (Total Evaluation Amount), "당일손익금액" (Intraday Profit/Loss Amount), and "당일손익율" (Intraday Profit/Loss Rate).
- Bottom Left:** A table showing stock information with columns: 종목코드, 종목명, 수량, 매수금, 현재가, 평가손익. The data includes:
 

|         |         |    |           |        |        |
|---------|---------|----|-----------|--------|--------|
| A000547 | 흥국화재2우B | 20 | 700,000   | 45,500 | 210000 |
| A000720 | 현대건설    | 20 | 1,480,000 | 69,300 | -94000 |
| A005030 | 부산주공    | 30 | 30,750    | 983    | -1260  |
| A005930 | 삼성전자    | 1  | 49,200    | 49,850 | 650    |
| A007390 | 네이처셀    | 10 | 305,500   | 29,850 | -7000  |
| A008250 | 미건산업    | 20 | 289,000   | 13,300 | -23000 |
- Bottom Right:** A summary table with tabs for "평입종목" (Average Purchase Stock), "이탈종목" (Exit Stock), and "주문기록" (Order Record). The "매도주문" tab is selected.

This screenshot shows the same Windows application window "Form1" after a user interaction. The interface remains largely the same, but the "주문기록" (Order Record) tab in the bottom-right summary table is now active, indicated by a red arrow pointing to it.

매도주문 요청은 SendOrder 함수를 호출해서 할 수 있습니다. 10.5장에서는 사용자의 주식 잔고에서 종목을 선택하고 매도주문을 요청합니다. 그래서 주식 잔고에서 종목이 선택 되었을 때 코드가 동작하도록 코드를 작성합니다.

Form1.cs의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 balanceDataGridView.SelectionChanged += dataGridViewSelectionChanged;
 axKHOpenAPI1.CommConnect();

}
```

추가한 dataGridViewSelectionChanged 함수의 내용을 작성합니다.

```
public void dataGridViewSelectionChanged(object sender, EventArgs e)
{
 if (sender.Equals(balanceDataGridView))
 {
 int rowIndex = balanceDataGridView.SelectedCells[0].RowIndex;
 if (balanceDataGridView.SelectedCells.Count > 0)
 {
 string[] currentPriceArray = balanceDataGridView["현재가", rowIndex].Value.ToString().Split(',');
 string stockCode = balanceDataGridView["종목코드", rowIndex].Value.ToString().Replace("A", "");
 string stockNumber = balanceDataGridView["수량", rowIndex].Value.ToString();
 string currentPrice = "";
 for (int i = 0; i < currentPriceArray.Length; i++)
 {
 currentPrice = currentPrice + currentPriceArray[i];
 }
 stockCodeLabel.Text = stockCode;
 orderPriceNumericUpDown.Value = long.Parse(currentPrice);
 orderNumberNumericUpDown.Value = long.Parse(stockNumber);
 }
 }
}
```

선택한 셀의 인덱스 정보를 먼저 가져옵니다. 선택된 행이 있다면 종목의 현재가, 수량, 종목코드 정보를 화면에 바인딩합니다. 거래 구분을 사용자가 선택하고 "매도주문" 버튼을 클릭한다면 매도주문이 요청 되도록 코드를 작성하면 됩니다. 종목코드의 경우에는 앞에 문자 "A"가 붙어있기 때문에 Replace 함수를 호출해서 문자열 "A"를 "" 공백으로 바꿔줍니다.

매도주문 버튼을 클릭했을 때 동작할 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 sellButton.Click += ButtonClicked;
 axKHOpenAPI1.CommConnect();

}
```

ButtonClicked 함수를 추가하고 if 문을 사용해서 sellButton에서 일어난 이벤트인 것을 구분합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 else if (sender.Equals(sellButton))
 {
 }
}
```

else if 문 안에서 종목코드, 거래구분, 주문가격, 주문수량 정보를 가져오고 SendOrder 함수를 호출해서 매도주문을 요청합니다.

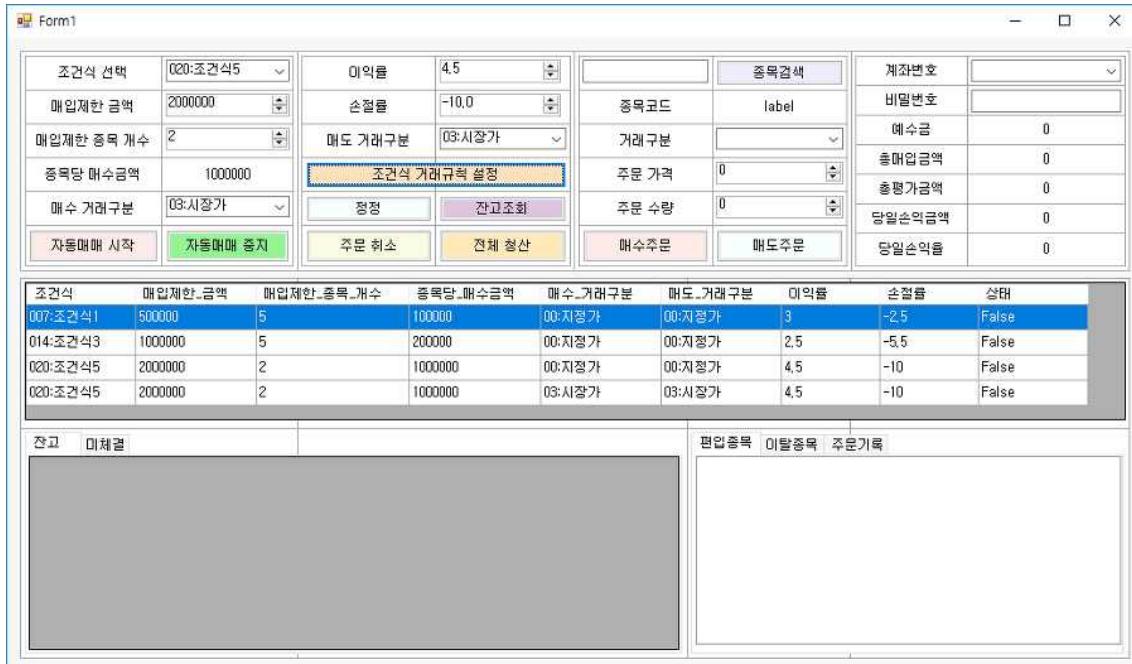
```
else if (sender.Equals(sellButton))
{
 string accountNumber = accountComboBox.Text;//계좌번호
 string stockCode = stockCodeLabel.Text;//종목코드
 string sellOrderType = orderComboBox.Text;//거래구분
 int sellOrderPrice = int.Parse(orderPriceNumericUpDown.Value.ToString());
 int sellOrderNumber = int.Parse(orderNumberNumericUpDown.Value.ToString());
```

```
if (accountNumber.Length>0 && sellOrderType.Length>0 &&
stockCode.Length>0 && sellOrderPrice>0 && sellOrderNumber>0)
{
 string[] orderType = sellOrderType.Split(':');
 axKHOpenAPI1.SendOrder("신규종목매도주문", "8289", accountNumber, 2,
stockCode, sellOrderNumber, sellOrderPrice, orderType[0], "");
}
```

계좌번호 , 종목코드 , 거래구분 , 가격 , 수량 값이 잘 들어 있다면 SendOrder 함수를 호출해서 매도주문을 요청합니다.

## 10.6 자동매매 거래규칙 설정

10.6 장에서는 자동매매 거래규칙을 설정합니다. "조건식 거래규칙 설정" 버튼을 클릭하면 거래규칙이 생성되도록 코드를 작성합니다. 거래규칙을 설정하기 전에 매입제한 금액, 매입제한 종목 개수를 설정했을 때 종목당 매수금액도 자동으로 계산되도록 합니다.



Form1.cs의 생성자에 "조건식 거래규칙 설정" 버튼을 클릭했을 때 동작할 함수를 추가합니다. 10장에서 버튼의 클릭에 관한 함수는 "ButtonClicked"입니다. 종목당 매수금액은 limitPriceNumericUpDown, limitNumberNumericUpDown에 함수를 추가해서 설정합니다.

```
public Form1()
{
 InitializeComponent();

 setAutoTradingRuleButton.Click += ButtonClicked;
 limitPriceNumericUpDown.ValueChanged += setBuyingPerStock;
 limitNumberNumericUpDown.ValueChanged += setBuyingPerStock;
 axKHOpenAPI1.CommConnect();
}
```

함

수의 내용을 차례대로 작성합니다. ButtonClicked 함수부터 내용을 작성합니다. if 문을 사용해서 요청 개체를 구분합니다.

```
public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(setAutoTradingRuleButton))
 {
 }
}
```

화면의 컨트롤로부터 값을 가져옵니다. 거래규칙을 설정하기 위해서 필요한 정보는 9가지입니다.

- 조건식
- 매입제한\_금액
- 매입제한\_종목\_개수
- 종목당\_매수금액
- 매수\_거래구분
- 매도\_거래구분
- 이익률
- 손절률
- 상태

if문 안에 코드를 작성하기 전에 클래스를 하나 만듭니다. 만들려는 클래스는 거래규칙을 모델링한 클래스입니다. 클래스의 이름을 AutoTradingRule이라고 명명합니다.

```
class AutoTradingRule
{
 public string 조건식이름;
 public long 매입제한_금액;
 public long 매입제한_종목_개수;
 public long 종목당_매수금액;
 public string 매수_거래구분;
 public string 매도_거래구분;
 public double 이익률;
 public double 손절률;
 public bool 상태;

 public AutoTradingRule(string conditionName, long limitBuyingStockPrice,
 long limitBuyingStockNumber, long limitBuyingPerStock, string
 autoBuyingOrderType, string autoSellingOrderType, double profitRate, double
 lossRate, bool status)
```

```
{
 this.조건식이름 = conditionName;
 this.매입제한_금액 = limitBuyingStockPrice;
 this.매입제한_종목_개수= limitBuyingStockNumber;
 this.종목당_매수금액 = limitBuyingPerStock;
 this.매수_거래구분= autoBuyingOrderType;
 this.매도_거래구분= autoSellingOrderType;
 this.이익률= profitRate;
 this.손절률= lossRate;
 this.상태 = status;
}
}
```

필요한 9가지 정보를 모두 클래스의 필드로 선언합니다. 사용자가 컨트롤에서 설정한 거래 규칙의 값을 모두 AutoTradingRule 객체에 저장하겠습니다. Form1.cs 의 클래스에 AutoTradingRule 리스트도 선언합니다.

```
public partial class Form1 : Form
{
 List<AutoTradingRule> autoTradingRuleList;
}
```

ButtonClicked 함수로 돌아와서 코드를 작성합니다.

```
if (sender.Equals(setAutoTradingRuleButton))
{
 string selectedCondition = conditionComboBox.Text;//조건식 선택
 long limitBuyingStockPrice = long.Parse(limitPriceNumericUpDown.Value.ToString());//매입제한 금액
 long limitBuyingStockNumber = long.Parse(limitNumberNumericUpDown.Value.ToString());//매입 제한 종목개수
 long limitBuyingPerStock = long.Parse(limitBuyingPerStockLabel.Text.ToString());//종목당 매수금액
 string autoBuyingOrderType = autoBuyOrderComboBox.Text;//매수 거래구분

 double profitRate = double.Parse(limitProfitRateNumericUpDown.Value.ToString());//이익률
 double lossRate = double.Parse(limitLossNumericUpDown.Value.ToString());//손절률
 string autoSellingOrderType = autoSellOrderComboBox.Text;//매도 거래구분
 bool status = false;//자동매매 진행여부
}
```

변수 status에는 현재 해당 조건식을 사용해서 자동매매를 진행하고 있는지를 표시한다. 값을 모두 저장했다면 화면에 바인딩하고 AutoTradingRule 리스트에 추가합니다.

```

public void ButtonClicked(object sender, EventArgs e)
{
 if (sender.Equals(setAutoTradingRuleButton))
 {
 string selectedCondition = conditionComboBox.Text;//조건식 선택
 long limitBuyingStockPrice = long.Parse(limitPriceNumericUpDown.Value.ToString());//매입제한 금액
 long limitBuyingStockNumber = long.Parse(limitNumberNumericUpDown.Value.ToString());//매입 제한 종목개수
 long limitBuyingPerStock = long.Parse(limitBuyingPerStockLabel.Text.ToString());//종목당 매수금액
 string autoBuyingOrderType = autoBuyOrderComboBox.Text;//매수 거래구분

 double profitRate = double.Parse(limitProfitRateNumericUpDown.Value.ToString());//이익률
 double lossRate = double.Parse(limitLossNumericUpDown.Value.ToString());//손절률
 string autoSellingOrderType = autoSellOrderComboBox.Text;//매도 거래구분
 bool status = false;//자동매매 진행여부

 if (selectedCondition.Length > 0 && limitBuyingStockPrice > 0 &&
 limitBuyingStockNumber > 0 &&
 limitBuyingPerStock > 0 && autoBuyingOrderType.Length > 0 &&
 profitRate > 0 &&
 lossRate != 0 && autoSellingOrderType.Length > 0)//값이 모두 잘 들어있었는지 체크
 {

 string[] conditionName = selectedCondition.Split(':');
 autoTradingRuleList.Add(new AutoTradingRule(conditionName[1],
 limitBuyingStockPrice, limitBuyingStockNumber,
 limitBuyingPerStock, autoBuyingOrderType, autoSellingOrderType,
 profitRate, lossRate, status));

 autoRuleDataGridView.Rows.Add();
 autoRuleDataGridView["거래규칙_조건식", autoRuleDataGridView.Rows.Count - 1].Value = selectedCondition;
 }
 }
}

```

```

 autoRuleDataGridView["거래규칙_매입제한_금액",
autoRuleDataGridView.Rows.Count - 1].Value = limitBuyingStockPrice;
 autoRuleDataGridView["거래규칙_매입제한_종목_개수",
autoRuleDataGridView.Rows.Count - 1].Value = limitBuyingStockNumber;
 autoRuleDataGridView["거래규칙_종목당_매수금액",
autoRuleDataGridView.Rows.Count - 1].Value = limitBuyingPerStock;
 autoRuleDataGridView["거래규칙_매수_거래구분",
autoRuleDataGridView.Rows.Count - 1].Value = autoBuyingOrderType;
 autoRuleDataGridView["거래규칙_매도_거래구분",
autoRuleDataGridView.Rows.Count - 1].Value = autoSellingOrderType;
 autoRuleDataGridView["거래규칙_이익률",
autoRuleDataGridView.Rows.Count - 1].Value = profitRate;
 autoRuleDataGridView["거래규칙_손절률",
autoRuleDataGridView.Rows.Count - 1].Value = lossRate;
 autoRuleDataGridView["거래규칙_상태",
autoRuleDataGridView.Rows.Count - 1].Value = status;

 MessageBox.Show("설정완료");
}

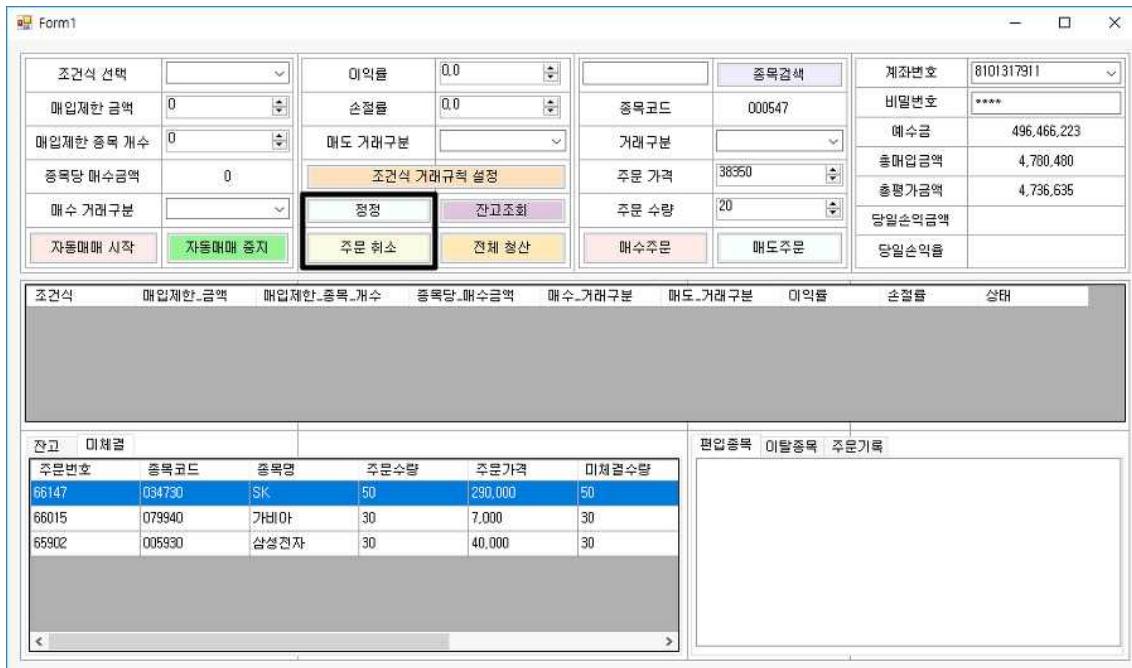
else if (selectedCondition.Length == 0 || limitBuyingStockPrice == 0 ||
limitBuyingStockNumber == 0 ||
limitBuyingPerStock == 0 || autoBuyingOrderType.Length == 0 ||
profitRate == 0 ||
lossRate == 0 || autoSellingOrderType.Length == 0)
{
 MessageBox.Show("거래규칙 값을 모두 입력하세요");
}
}

```

else if 문을 사용해서 변수의 값이 1개라도 들어있지 않으면 "거래규칙 값을 모두 입력하세요"라는 메세지 박스가 출력되도록 코드를 작성했습니다.

## 10.7 주문 정정 , 취소

10.7장에서는 주문 정정 , 취소 기능을 구현하도록 하겠습니다. 미체결 DataGridView에서 미체결 주문을 선택 했을때 정정 , 취소 요청을 보낼 수 있도록 코드를 작성하겠습니다.



주문 취소는 미체결 DataGridView에서 항목을 선택하고 클릭해서 바로 동작하게 코드를 작성하면 됩니다. 하지만 주문 정정은 가격 , 수량 , 거래구분 정보를 변경 할 수 있어야 합니다. 미체결 DataGridView에서 선택한 항목이 있다면 화면에 선택한 항목 정보를 바인딩해서 주문을 정정 할 수 있도록 합니다.

Form1.cs의 생성자에 함수를 추가합니다.

```
public Form1()
{
 InitializeComponent();

 outstandingDataGridView.SelectionChanged += dataGridSelectionChanged;
 orderFixButton.Click += ButtonClicked;
 orderCancelButton.Click += ButtonClicked;
 axKHOpenAPI1.CommConnect();

}
```

dataGridViewSelectionChanged 함수부터 내용을 작성합니다.

```
public void dataGridViewSelectionChanged(object sender, EventArgs e)
{
 if (sender.Equals(balanceDataGridView))
 {

 }
 else if (sender.Equals(outstandingDataGridView))
 {

 }
}
```

if문을 사용해서 balanceDataGridView의 동작과 구분합니다.

```
else if (sender.Equals(outstandingDataGridView))
{
 if (outstandingDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = outstandingDataGridView.SelectedCells[0].RowIndex;
 string[] outstandingPriceArray = outstandingDataGridView["주문가격",
rowIndex].Value.ToString().Split(',');
 string outstandingStockCode = outstandingDataGridView["종목코드",
rowIndex].Value.ToString();
 string outstandingStockNumber = outstandingDataGridView["미체결수량",
rowIndex].Value.ToString();
 string outstandingPrice = "";
 for (int i = 0; i < outstandingPriceArray.Length; i++)
 {
 outstandingPrice = outstandingPrice + outstandingPriceArray[i];
 }
 stockCodeLabel.Text = outstandingStockCode;
 orderPriceNumericUpDown.Value = long.Parse(outstandingPrice);
 orderNumberNumericUpDown.Value
 =
long.Parse(outstandingStockNumber);
 }
}
```

선택된 셀의 rowIndex 를 가져오고 미체결 DataGridView의 정보를 화면에 바인딩 합니다. 이제 앞서 추가했었던 ButtonClicked 함수를 추가하고 함수의 내용을 작성합니다. if문을 사용해서 orderFixButton 과 orderCancelButton 을 구분하면 됩니다.

```
else if (sender.Equals(orderFixButton))
{
}
else if (sender.Equals(orderCancelButton))
{
}
```

orderFixButton 부분 먼저 작성합니다.

```
else if (sender.Equals(orderFixButton))
{
 string accountNubmer = accountComboBox.Text;
 if (outstandingDataGridView.SelectedCells.Count>0 &&
accountNubmer.Length>0)
 {
 }
}
```

주문을 정정하려면 계좌번호가 필요합니다. 그리고 미체결 DataGridView에서 선택된 항목이 있어야 주문번호를 알 수 있습니다. 해당 항목이 들어있는지 체크합니다. 항목을 체크하고 화면으로부터 값을 가져옵니다. 주문을 정정하기 위해서는 6가지 항목이 필요합니다.

- 주문구분
- 종목코드
- 거래구분
- 주문수량
- 주문가격
- 주문번호

```
else if (sender.Equals(orderFixButton))
{
 string accountNubmer = accountComboBox.Text;
 if (outstandingDataGridView.SelectedCells.Count>0 &&
accountNubmer.Length>0)
```

```
{
 int rowIndex = outstandingDataGridView.SelectedCells[0].RowIndex;
 string orderType = outstandingDataGridView["주문구분",
rowIndex].Value.ToString(); //주문구분
 string tradingType = orderComboBox.Text; //거래구분
 string stockCode = outstandingDataGridView["종목코드",
rowIndex].Value.ToString(); //종목코드
 int orderNumber =
int.Parse(orderNumberNumericUpDown.Value.ToString()); //주문수량
 int orderPrice = int.Parse(orderPriceNumericUpDown.Value.ToString()); //주문가격
 string orderCode = outstandingDataGridView["주문번호",
rowIndex].Value.ToString(); //주문번호
}
}
```

if문을 사용해서 값을 체크하고 다음 내용을 작성합니다.

```
else if (sender.Equals(orderFixButton))
{
 string accountNubmer = accountComboBox.Text;
 if (outstandingDataGridView.SelectedCells.Count > 0 &&
accountNubmer.Length > 0)
 {
 int rowIndex = outstandingDataGridView.SelectedCells[0].RowIndex;
 string orderType = outstandingDataGridView["주문구분",
rowIndex].Value.ToString(); //주문구분
 string tradingType = orderComboBox.Text; //거래구분
 string stockCode = outstandingDataGridView["종목코드",
rowIndex].Value.ToString(); //종목코드
 int orderNumber =
int.Parse(orderNumberNumericUpDown.Value.ToString()); //주문수량
 int orderPrice = int.Parse(orderPriceNumericUpDown.Value.ToString()); //주문가격
 string orderCode = outstandingDataGridView["주문번호",
rowIndex].Value.ToString(); //주문번호
 }
 if (orderType.Length > 0 && tradingType.Length > 0 && stockCode.Length > 0 &&
orderNumber > 0 && orderPrice > 0 && orderCode.Length > 0){
 }
}
```

---

orderType 변수에 들어있는 매수, 매도 값을 구분해서 주문 정정을 요청합니다.

```

else if (sender.Equals(orderFixButton))
{
 string accountNubmer = accountComboBox.Text;
 if (outstandingDataGridView.SelectedCells.Count>0 &&
accountNubmer.Length>0)
 {
 int rowIndex = outstandingDataGridView.SelectedCells[0].RowIndex;
 string orderType = outstandingDataGridView["주문구분",
rowIndex].Value.ToString();
 string tradingType = orderComboBox.Text;
 string stockCode = outstandingDataGridView["종목코드",
rowIndex].Value.ToString();
 int orderNumber = int.Parse(orderNumberNumericUpDown.Value.ToString());
 int orderPrice = int.Parse(orderPriceNumericUpDown.Value.ToString());
 string orderCode = outstandingDataGridView["주문번호",
rowIndex].Value.ToString();

 if (orderType.Length>0 && tradingType.Length>0 && stockCode.Length>0
&& orderNumber>0 && orderPrice>0 && orderCode.Length>0)
 {
 string[] tradingTypeArray = tradingType.Split(':');
 if (orderType=="-매도")
 {
 axKHOpenAPI1.SendOrder("종목주문정정", "1430", accountNubmer,
6, stockCode, orderNumber, orderPrice, tradingTypeArray[0], orderCode);
 MessageBox.Show("정정요청 완료");
 }
 else if (orderType=="+매수")
 {
 axKHOpenAPI1.SendOrder("종목주문정정", "1430", accountNubmer,
5, stockCode, orderNumber, orderPrice, tradingTypeArray[0], orderCode);
 MessageBox.Show("정정요청 완료");
 }
 }
 }
}

```

주문 취소 요청도 코드를 작성하겠습니다.

```

else if (sender.Equals(orderCancelButton))
{
 if (outstandingDataGridView.SelectedCells.Count > 0)
 {
 string accountNumber = accountComboBox.Text;
 if (outstandingDataGridView.SelectedCells.Count > 0 &&
accountNumber.Length > 0)
 {
 int rowIndex = outstandingDataGridView.SelectedCells[0].RowIndex;
 string orderType = outstandingDataGridView["주문구분", rowIndex].Value.ToString();
 string tradingType = orderComboBox.Text;
 string stockCode = outstandingDataGridView["종목코드", rowIndex].Value.ToString();
 int orderNumber = int.Parse(orderNumberNumericUpDown.Value.ToString());
 int orderPrice = int.Parse(orderPriceNumericUpDown.Value.ToString());
 string orderCode = outstandingDataGridView["주문번호", rowIndex].Value.ToString();
 }
 }
}

```

```
 if (orderType.Length > 0 && tradingType.Length > 0 &&
stockCode.Length > 0 && orderNumber > 0 && orderPrice > 0 &&
orderCode.Length > 0)
{
 string[] tradingTypeArray = tradingType.Split(':');
 if (orderType == "-매도")
 {
 axKHOpenAPI1.SendOrder("종목주문정정", "1430",
accountNubmer, 4, stockCode, orderNumber, orderPrice, tradingTypeArray[0],
orderCode);
 MessageBox.Show("취소요청 완료");
 }
 else if (orderType == "+매수")
 {
 axKHOpenAPI1.SendOrder("종목주문정정", "1430",
accountNubmer, 3, stockCode, orderNumber, orderPrice, tradingTypeArray[0],
orderCode);
 MessageBox.Show("취소요청 완료");
 }
}
}
```

정정 요청 코드와 동일합니다. SendOrder함수에 작성하는 인자 값만 다릅니다.

## 10.8 자동매매 시작

10.8장에서는 자동매매 시작 버튼을 클릭했을 때 동작할 코드를 작성하겠습니다. 10.6장에서 자동매매 거래 규칙을 설정하는 방법에 대해서 알아봤습니다. 10.6장에서 설정한 자동매매 거래 규칙에 따라서 자동매매를 진행하도록 코드를 작성하는 것입니다.



```
public Form1()
{
 InitializeComponent();

 setAutoTradingRuleButton.Click += ButtonClicked;
 autoTradingStartButton.Click += ButtonClicked;

}
```

Form1.cs의 생성자에 ButtonClicked 함수를 추가합니다. ButtonClicked함수의 내용을 작성합니다. 자동매매는 autoRuleDataGridView에서 거래 규칙을 선택하고 진행하게 만들 것입니다.

먼저 autoRuleDataGridView에서 선택된 항목이 있는지 체크합니다.

```
else if (sender.Equals(autoTradingStartButton))
{
 if (autoRuleDataGridView.SelectedCells.Count>0)
 {
 }
}
```

거래규칙을 선택하고 autoTradingStartButton 을 클릭하면 상태를 True로 변경하고 조건 검색을 요청합니다. 조건 식을 사용해서 검색되는 종목을 설정한 매입제한 금액에 맞춰서 매수합니다. 조건식으로 검색되는 종목들을 알아내기 위해서 SendCondition 함수를 호출합니다.

```
else if (sender.Equals(autoTradingStartButton))
{
 if (autoRuleDataGridView.SelectedCells.Count>0)
 {
 int rowIndex = autoRuleDataGridView.SelectedCells[0].RowIndex;
 autoRuleDataGridView["거래규칙_상태", rowIndex].Value = true;

 string autoTradingCondition = autoRuleDataGridView["거래규칙_조건식",
rowIndex].Value.ToString();
 string[] autoTradingArray = autoTradingCondition.Split(':');
 autoSreenNumber++;
 string scrNumber = autoSreenNumber.ToString();

 axKHOpenAPI1.SendCondition(scrNumber,autoTradingArray[1],int.Parse(autoTradingArray[0]),0);//종목검색

 axKHOpenAPI1.SendCondition(scrNumber,autoTradingArray[1],int.Parse(autoTradingArray[0]),1);//실시간 편입종목검색
 }
}
```

SendCondition 함수를 호출했다면 OnReceiveTrCondition 함수에서 종목 정보를 요청 할 수 있습니다.

```
public Form1()
{
 InitializeComponent();

 axKHOpenAPI1.OnReceiveTrCondition += onReceiveTrCondition;

}
```

Form1.cs의 생성자에 함수를 추가하고 내용을 작성합니다.

```
public void onReceiveTrCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrConditionEvent e)
{
 if (e.strCodeList.Length>0)
 {
 string stockCodeList = e.strCodeList.Remove(e.strCodeList.Length - 1);
 int stockCount = stockCodeList.Split(';').Length;
 if (stockCount<=100)
 {
 axKHOpenAPI1.CommKwRqData(stockCodeList, 0, stockCount, 0, "조건
검색종목", "5100");
 }
 if (e.nNext != 0)
 {
 axKHOpenAPI1.SendCondition("5101", e.strConditionName, e.nIndex,
1);
 }
 }
 else if (e.strCodeList.Length==0)
 {
 MessageBox.Show("검색된 종목이 없습니다.");
 }
}
```

검색된 종목의 코드리스트의 길이를 확인하고 100 종목 이하라면 CommKwRqData 함수를 호출해서 종목을 검색합니다. e.nNext의 값을 체크해서 검색되는 종목이 100개 이상이라면 SendCondition 함수를 다시 호출합니다. 9.3장의 내용을 참고해서 작성하면 됩니다. CommKwRqData 함수를 호출하면 OnReceiveTrData 함수가 호출됩니다. OnReceiveTrData 함수를 사용해서 검색한 종목들의 정보를 가져옵니다.

```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 else if (e.sRQName == "조건검색종목")
 {
 }
}

```

onReceiveTrData에서는 이미 많은 요청들을 처리하고 있습니다. else if 문을 사용해서 요청을 구분해줍니다. 아래와 같이 코드를 작성해서 종목 정보를 요청합니다. 9.3장에서 이미 종목 정보를 요청하는 것을 해봤습니다. 10.8장이 9.3장과 다른 점은 종목 정보를 요청하고 매수 주문을 요청하는 것입니다. 검색되는 종목을 제한 금액 안에서 계속 매수 주문을 요청합니다.

```

else if (e.sRQName == "조건검색종목")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);

 for (int i = 0; i < count; i++)
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName,
i, "종목코드").Trim();
 }
}

```

**종목당 매수금액보다 현재 종목 가격이 더 클때 (매수가능)**

현재 종목을 종목당 매수금액만큼 매수

**예1) 종목당 매수 금액이 현재 가격으로 나누어 떨어지는 경우**

종목당 매수금액 50,000 원

종목의 현재 가격 5,000원

->  $50,000 \text{ (종목당 매수금액)} / 5,000 \text{ (종목의 현재가격)} = 10 \text{ (종목 주문 수량)}$

**예2) 종목당 매수 금액이 현재 가격으로 나누어 떨어지지 않는 경우**

종목당 매수금액 50,000원

종목의 현재 가격 7,680원

->  $50,000\text{원 (종목당 매수금액)} / 7,680 \text{ (종목의 현재 가격)} = 6.5104\dots \rightarrow \text{int형으로 변환되면 } 6 \text{ (종목 주문 수량)}$

---

종목당 매수 금액이 현재 종목 가격보다 작을때 (매수 불가능)

종목당 매수 금액 50,000원

종목의 현재 가격 300,000원

**1개도 매수 할 수 없는 상태**

---

종목당 매수 금액이 현재 가격보다 큰 경우를 if 문으로 처리해서 매수 주문을 요청합니다.

```
else if (e.sRQName == "조건검색종목")
{
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName); //조건식으로
 검색되는 종목의 개수

 if (autoRuleDataGridView.SelectedCells.Count>0) //선택된 DataGridView 셀을 체
 크
 {
 int rowIndex = autoRuleDataGridView.SelectedCells[0].RowIndex;

 int autoTradingRuleID = int.Parse(autoRuleDataGridView["거래규칙_번호",
rowIndex].Value.ToString());
 int autoOrderPricePerStock = int.Parse(autoRuleDataGridView["거래규칙_종
목당_매수금액", rowIndex].Value.ToString());
 int autoTradingLimitOrderNumber = int.Parse(autoRuleDataGridView["거래
규칙_매입제한_종목_개수", rowIndex].Value.ToString());

 string autoBuyOrderType = autoRuleDataGridView["거래규칙_매수_거래구분",
rowIndex].Value.ToString();
 string[] autoBuyOrderArray = autoBuyOrderType.Split(':');

 string autoRuleStatus = autoRuleDataGridView["거래규칙_상태",
rowIndex].Value.ToString();

 int autoRuleListIndex = autoTradingRuleList.FindIndex(o => o.번호 ==
autoRuleID);

 if (autoRuleStatus == "시작" && accountComboBox.Text.Length>0) //거래규
칙 상태가 "시작"이면
```

```

 {
 ACCOUNT_NUMBER = accountComboBox.Text;
 for (int i = 0; i < count; i++)
 {
 if (i > autoTradingLimitOrderNumber)//제한 종목개수 초과하면
break;
 {
 break;
 }
 else
 {
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목코드").Trim();
 string stockName = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "종목명").Trim();
 int stockPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i, "현재가"
").Replace("-", ""));
 if (autoOrderPricePerStock > stockPrice)
 {
 int orderNumber = autoOrderPricePerStock / stockPrice;
 axKHOpenAPI1.SendOrder("자동거래매수주문", "5149",
ACCOUNT_NUMBER, 1, stockCode, orderNumber, stockPrice,
autoBuyOrderArray[0], "");
 autoTradingRuleList[autoRuleListIndex].autoTradingPurchaseStockList.Add(new
AutoTradingPurchaseStock(stockCode, stockPrice, 0));
 }
 }
 }
 }
}

```

만약 매입 제한 종목 개수가 5개인데 조건식으로 검색되는 종목이 3개 밖에 되지 않는 경우 매입 제한 개수만큼 모두 매수하지 못했다면 OnReceiveRealCondition 함수 안에서 종목이 편입 되었을때 매수 주문을 요청하도록 합니다.

```

public void onReceiveRealCondition(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealConditionEvent e)
{
 if (e.strType=="I")
 {
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 insertListBox.Items.Add("종목편입| 조건인덱스 : " + e.strConditionIndex + " | 종목코드 : " + e.sTrCode + " | " + "종목명 : " + stockName);
 currentCondition = e.strConditionIndex+":"+e.strConditionName;
 axKHOpenAPI1.SetInputValue("종목코드", e.sTrCode);
 axKHOpenAPI1.CommRqData("자동거래2차매수", "opt10001", 0, "5152");
 }
 else if (e.strType=="D")
 {
 string stockName = axKHOpenAPI1.GetMasterCodeName(e.sTrCode);
 deleteListBox.Items.Add("종목이탈| 조건인덱스 : " + e.strConditionIndex + " | 종목코드 : " + e.sTrCode + " | " + "종목명 : " + stockName);
 }

}

(OnReceiveTrData 함수)
else if (e.sRQName=="자동거래2차매수")
{
 int stockPrice = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "현재가").Replace("-", ""));
 string stockCode = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, 0, "종목코드").Trim();
 int ruleIndex = autoTradingRuleList.FindIndex(o => o.조건식이름 == currentCondition);

 if (autoTradingRuleList[ruleIndex].autoTradingPurchaseStockList.Count < autoTradingRuleList[ruleIndex].매입제한_종목_개수 && autoTradingRuleList[ruleIndex].종목당_매수금액 > stockPrice)
}

```

```
{
 if (accountComboBox.Text.Length > 0 && autoTradingRuleList[ruleIndex].상태 == "시작")
 {
 int autoOrderPricePerStock = autoTradingRuleList[ruleIndex].종목당_매수금액;
 int orderNumber = autoOrderPricePerStock / stockPrice;
 ACCOUNT_NUMBER = accountComboBox.Text;
 string[] autoOrderTypeArray = autoTradingRuleList[ruleIndex].매수_거래구분.Split(':');

 axKHOpenAPI1.SendOrder("자동거래2차매수주문", "5154",
ACCOUNT_NUMBER, 1, stockCode, orderNumber, stockPrice,
autoOrderTypeArray[0], "");

 autoTradingRuleList[ruleIndex].autoTradingPurchaseStockList.Add(new
AutoTradingPurchaseStock(stockCode, stockPrice, 0));
 }
}
}
```

매도 주문은 OnReceiveRealData 함수 안에서 종목의 실시간 가격 변화에 따라 처리합니다.

```
public void onReceiveRealData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveRealDataEvent e)
{
 for (int i = 0; i < autoTradingRuleList.Count; i++)
 {
 if (autoTradingRuleList[i].상태=="시작")
 {
 double profitRate = autoTradingRuleList[i].이익률*0.01;
 double lossRate = autoTradingRuleList[i].손절률*0.01;

 for (int j = 0; j < autoTradingRuleList[i].autoTradingPurchaseStockList.Count; j++)
 {

 autoTradingRuleList[i].autoTradingPurchaseStockList[j].currentPrice =
int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealKey, 10));
 }
 }
 }
}
```

```

autoTradingRuleList[i].autoTradingPurchaseStockList[j].boughtCount
int.Parse(axKHOpenAPI1.GetCommRealData(e.sRealKey,930));

 string stockCode
autoTradingRuleList[i].autoTradingPurchaseStockList[j].stockCode;
 int currentPrice
autoTradingRuleList[i].autoTradingPurchaseStockList[j].currentPrice;
 int boughtPrice
autoTradingRuleList[i].autoTradingPurchaseStockList[j].boughtPrice;
 int boughtCount
autoTradingRuleList[i].autoTradingPurchaseStockList[j].boughtCount;
 string orderType = autoTradingRuleList[i].매도_거래구분;
 string[] orderTypeArray = orderType.Split(':');

 if (currentPrice == (boughtPrice+(boughtPrice*profitRate)))
{
 axKHOpenAPI1.SendOrder("이익을매도주문", "8889",
ACCOUNT_NUMBER, 2, stockCode, boughtCount, currentPrice,
orderTypeArray[0], "");

}
 else if (currentPrice == (boughtPrice - (boughtPrice * profitRate)))
{
 axKHOpenAPI1.SendOrder("손절을매도주문", "8789",
ACCOUNT_NUMBER, 2, stockCode, boughtCount, currentPrice,
orderTypeArray[0], "");

}

}
}
}
}

```

전체코드

전체코드는 GitHub에서 확인 할 수 있습니다.

Form1

|            |          |             |        |       |        |            |             |
|------------|----------|-------------|--------|-------|--------|------------|-------------|
| 조건식 선택     | 020:조건식5 | 미익률         | 2.0    | 증목검색  | 계좌번호   | 8101317911 |             |
| 매입제한 금액    | 1000000  | 순절률         | -2.0   | 증목코드  | 000547 | 비밀번호       | ****        |
| 매입제한 증목 개수 | 5        | 매도 거래구분     | 00:지정가 | 거래 구분 | 00:지정가 | 예수금        | 496,466,223 |
| 증목당 매수금액   | 200000   | 조건식 거래규칙 설정 |        | 주문 가격 | 35200  | 총매입금액      | 5,582,400   |
| 매수 거래구분    | 00:지정가   | 정정          | 잔고조회   | 주문 수량 | 20     | 총평가금액      | 5,362,785   |
| 자동매매 시작    | 자동매매 중지  | 주문 취소       | 전체 청산  | 매수주문  | 매도주문   | 당일순익금액     | 15,263      |
|            |          |             |        |       |        | 당일순익률      | 1.58        |

| 번호 | 조건식      | 매입제한_금액 | 매입제한_증목_개수 | 증목당_매수금액 | 매수_거래구분 | 매도_거래구분 | 미익률 | 순절률 | 상태 |
|----|----------|---------|------------|----------|---------|---------|-----|-----|----|
| 1  | 015:조건식4 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |
| 2  | 014:조건식3 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 시작 |
| 3  | 007:조건식1 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |
| 4  | 020:조건식5 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |

| 잔고      | 마체결     | 증목코드 | 증목명       | 수량     | 매수금     | 현재가 | 평가손익 |
|---------|---------|------|-----------|--------|---------|-----|------|
| A000547 | 한국화재2우B | 20   | 700,000   | 35,200 | 4000    |     |      |
| A000720 | 현대건설    | 20   | 1,480,000 | 68,900 | -102000 |     |      |
| A001500 | 현대차투자증권 | 17   | 198,050   | 11,650 | 0       |     |      |
| A005030 | 부산주공    | 30   | 30,750    | 976    | -1470   |     |      |
| A006345 | 대원건설우   | 26   | 197,600   | 7,600  | 0       |     |      |
| A007390 | 내이처셀    | 10   | 305,500   | 29,550 | -10000  |     |      |

편입증목 : 한국화재2우B | 주문수량 : 20  
 거래구분 : 매도  
 주문번호 : 00630883 | 주문상태 : 체결  
 증목명 : 한국화재2우B | 주문수량 : 20  
 거래구분 : 매도  
 주문번호 : 00630883 | 주문상태 : 체결  
 증목명 : 한국화재2우B | 주문수량 : 20  
 거래구분 : 매도

Form1

|            |          |             |        |       |        |            |             |
|------------|----------|-------------|--------|-------|--------|------------|-------------|
| 조건식 선택     | 020:조건식5 | 미익률         | 2.0    | 증목검색  | 계좌번호   | 8101317911 |             |
| 매입제한 금액    | 1000000  | 순절률         | -2.0   | 증목코드  | 000547 | 비밀번호       | ****        |
| 매입제한 증목 개수 | 5        | 매도 거래구분     | 00:지정가 | 거래 구분 | 00:지정가 | 예수금        | 496,466,223 |
| 증목당 매수금액   | 200000   | 조건식 거래규칙 설정 |        | 주문 가격 | 35200  | 총매입금액      | 5,582,400   |
| 매수 거래구분    | 00:지정가   | 정정          | 잔고조회   | 주문 수량 | 20     | 총평가금액      | 5,362,785   |
| 자동매매 시작    | 자동매매 중지  | 주문 취소       | 전체 청산  | 매수주문  | 매도주문   | 당일순익금액     | 15,263      |
|            |          |             |        |       |        | 당일순익률      | 1.58        |

| 번호 | 조건식      | 매입제한_금액 | 매입제한_증목_개수 | 증목당_매수금액 | 매수_거래구분 | 매도_거래구분 | 미익률 | 순절률 | 상태 |
|----|----------|---------|------------|----------|---------|---------|-----|-----|----|
| 1  | 015:조건식4 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |
| 2  | 014:조건식3 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 시작 |
| 3  | 007:조건식1 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |
| 4  | 020:조건식5 | 1000000 | 5          | 200000   | 00:지정가  | 00:지정가  | 2   | -2  | 정지 |

| 잔고      | 마체결     | 증목코드 | 증목명       | 수량     | 매수금     | 현재가 | 평가손익 |
|---------|---------|------|-----------|--------|---------|-----|------|
| A000547 | 한국화재2우B | 20   | 700,000   | 35,200 | 4000    |     |      |
| A000720 | 현대건설    | 20   | 1,480,000 | 68,900 | -102000 |     |      |
| A001500 | 현대차투자증권 | 17   | 198,050   | 11,650 | 0       |     |      |
| A005030 | 부산주공    | 30   | 30,750    | 976    | -1470   |     |      |
| A006345 | 대원건설우   | 26   | 197,600   | 7,600  | 0       |     |      |
| A007390 | 내이처셀    | 10   | 305,500   | 29,550 | -10000  |     |      |

편입증목 : 현대차투자증권 | 주문수량 : 17  
 거래구분 : 매수  
 주문번호 : 00631114 | 주문상태 : 접수  
 증목명 : 현대차투자증권 | 주문수량 : 16  
 거래구분 : 매수  
 주문번호 : 00631114 | 주문상태 : 접수  
 증목명 : 현대차투자증권 | 주문수량 : 16  
 거래구분 : 매수  
 주문번호 : 00631115 | 주문상태 : 접수  
 증목명 : 현대차투자증권 | 주문수량 : 71  
 거래구분 : 매수  
 주문번호 : 00631115 | 주문상태 : 접수  
 증목명 : 현대차투자증권 | 주문수량 : 71  
 거래구분 : 매수

## 10.9 자동매매 중지 , 전체 청산

10.9장에서는 자동매매 중지 , 전체청산 버튼을 클릭했을때 동작할 코드를 작성합니다.



전체 청산

```
public Form1()
{

 InitializeComponent();
 autoTradingStopButton.Click += ButtonClicked;

}
```

자동매매 중지 버튼인 autoTradingStopButton을 클릭했을때 ButtonClicked 함수가 동작하도록 합니다. ButtonClicked 함수에서는 거래규칙의 상태를 "정지"로 변경해서 더이상 자동매수가 진행되지 못하게 합니다. 아래 코드는 ButtonClicked 함수의 내용중 일부입니다. autoTradingStopButton이 클릭되면 선택된 행의 거래 규칙 상태를 변경합니다.

```
else if (sender.Equals(autoTradingStopButton))
{
 if (autoRuleDataGridView.SelectedCells.Count > 0)
 {
 int rowIndex = autoRuleDataGridView.SelectedCells[0].RowIndex;
```

```

 autoRuleDataGridView["거래규칙_상태", rowIndex].Value = "정지";

}
}

```

다음은 전체 청산 버튼입니다. 전체 청산 버튼을 클릭 했을때는 autoTradingRuleList에서 구매한 autoTradingPurchaseList의 모든 종목을 매도하도록 합니다. SellAllStockButton이 클릭되면 가지고 있는 모든 주식을 시장가로 처분합니다.

```

else if (sender.Equals(SellAllStockButton))
{
 for (int i = 0; i < autoTradingRuleList.Count; i++)
 {
 for (int j = 0; j < autoTradingRuleList[i].autoTradingPurchaseStockList.Count; j++)
 {
 string stockCode = autoTradingRuleList[i].autoTradingPurchaseStockList[j].stockCode;
 int boughtCount = autoTradingRuleList[i].autoTradingPurchaseStockList[j].boughtCount;
 int currentPrice = autoTradingRuleList[i].autoTradingPurchaseStockList[j].boughtPrice;
 axKHOpenAPI1.SendOrder("전체청산주문", "9999", ACCOUNT_NUMBER,
2, stockCode, boughtCount, currentPrice, "03", "");
 }
 }
}

```

SendOrder 함수를 호출해서 가지고 있는 모든 주식을 매도하고 있습니다.

## 부록 A : 기타 API 함수 Preview

11장에서는 주식 종목의 차트를 프로그램에서 그려봅니다. 차트에서 드러나는 정보를 기반으로 매수, 매도를 하고 있다면 유익한 내용이 될 것입니다.

### Contents

11.1장에서는 OnReceiveTrData 함수를 사용해서 프로그램에서 차트를 그려봅니다. 차트의 Series 속성을 활용하는 방법을 알아봅니다.

## 차트 그리기

11장에서는 주식 가격 차트를 프로그램에서 그려보도록 하겠습니다.



Form1.cs의 생성자에 함수를 추가합니다. 11장의 코드 작성 순서는 다음과 같습니다.

- 로그인
- OnEventConnect 함수
- 차트 출력 버튼 클릭 함수

```
public partial class Form1 : Form
{
 List<stockInfo> stockList;
 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect(); //로그인
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 }
}
```

로그인 함수를 작성하고 OnEventConnect 함수를 작성합니다.

```
public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
```

```

{

 stockList = new List<stockInfo>();

 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new
 stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));

 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
}

.....
.....
.....
class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode, string stockName)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}
class priceInfoObject
{
 public string 일자;
 public int 시가;
 public int 고가;
 public int 저가;
 public int 종가;
 public priceInfoObject(string 일자 , int 시가 , int 고가 , int 저가 , int 종가)
}

```

```
{
 this.일자 = 일자;
 this.시가 = 시가;
 this.고가 = 고가;
 this.저가 = 저가;
 this.종가 = 종가;
}
}
```

이제 차트 출력 버튼을 클릭했을 때 동작할 함수를 작성합니다.

```
public Form1()
{
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 chartPlotButton.Click += chartPlot;
}
```

함수의 내용을 작성합니다.

```
public void chartPlot(object sender, EventArgs e)
{
 if (sender.Equals(chartPlotButton))
 {
 priceInfoList = new List<priceInfoObject>();
 if (stockTextBox.Text.Length > 0)
 {
 int index = stockList.FindIndex(o => o.stockName == stockTextBox.Text);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.SetInputValue("기준일자", DateTime.Now.ToString("yyyyMMdd"));
 axKHOpenAPI1.SetInputValue("수정주가구분", "1");
 axKHOpenAPI1.CommRqData("주식일봉차트조회", "opt10081", 0, "8100");

 }
 }
}
```

현재 11장에서 사용하려는 TR의 번호는 opt10081 "주식 일봉차트 조회"입니다.

onReceiveTrData 함수에서 실질적인 그래프를 그려줍니다.

```

public void onReceiveTrData(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
{
 if (e.sRQName=="주식일봉차트조회")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);

 for (int i = 0; i < count; i++)
 {
 string 일자 = axKHOpenAPI1.GetCommData(e.sTrCode, e.sRQName, i,
"일자").Trim();
 int 시가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "시가").Trim());
 int 고가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "고가").Trim());
 int 저가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "저가").Trim());
 int 종가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Trim());

 priceInfoList.Add(new priceInfoObject(일자, 시가, 고가, 저가, 종가));

 stockChart.Series["Series1"].Points.AddXY(priceInfoList[i].일자,
priceInfoList[i].고가);
 stockChart.Series["Series1"].Points[i].YValues[1] = priceInfoList[i].저가;
 stockChart.Series["Series1"].Points[i].YValues[2] = priceInfoList[i].시가;
 stockChart.Series["Series1"].Points[i].YValues[3] = priceInfoList[i].종가;
 }
 }
}

```

## 전체코드

```

using System;
using System.Collections.Generic;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Forms;

namespace WindowsFormsApp13
{
 public partial class Form1 : Form
 {
 List<stockInfo> stockList;
 List<priceInfoObject> priceInfoList;
 Series chartSeries;

 public Form1()
 {
 InitializeComponent();
 axKHOpenAPI1.CommConnect();
 axKHOpenAPI1.OnEventConnect += onEventConnect;
 chartPlotButton.Click += chartPlot;
 axKHOpenAPI1.OnReceiveTrData += onReceiveTrData;

 chartSeries = stockChart.Series["Series1"];
 stockChart.Series["Series1"]["PriceUpColor"] = "Red";
 stockChart.Series["Series1"]["PriceDownColor"] = "Blue";
 }

 public void onReceiveTrData(object sender ,
AxKHOpenAPILib._DKHOpenAPIEvents_OnReceiveTrDataEvent e)
 {
 if (e.sRQName=="주식일봉차트조회")
 {
 int count = axKHOpenAPI1.GetRepeatCnt(e.sTrCode, e.sRQName);

 for (int i = 0; i < count; i++)
 {
 string 일자 = axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "일자").Trim();
 int 시가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "시가").Trim());
 }
 }
 }
 }
}

```

```

 int 고가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "고가").Trim());
 int 저가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "저가").Trim());
 int 종가 = int.Parse(axKHOpenAPI1.GetCommData(e.sTrCode,
e.sRQName, i, "현재가").Trim());

 priceInfoList.Add(new priceInfoObject(일자, 시가, 고가, 저가, 종
가));

 stockChart.Series["Series1"].Points.AddXY(priceInfoList[i].일자,
priceInfoList[i].고가);
 stockChart.Series["Series1"].Points[i].YValues[1] =
priceInfoList[i].저가;
 stockChart.Series["Series1"].Points[i].YValues[2] =
priceInfoList[i].시가;
 stockChart.Series["Series1"].Points[i].YValues[3] =
priceInfoList[i].종가;

 }

}

}

public void chartPlot(object senrder, EventArgs e)
{
 if (senrder.Equals(chartPlotButton))
 {
 priceInfoList = new List<priceInfoObject>();
 if (stockTextBox.Text.Length > 0)
 {
 int index = stockList.FindIndex(o => o.stockName == stockTextBox.Text);
 string stockCode = stockList[index].stockCode;
 axKHOpenAPI1.SetInputValue("종목코드", stockCode);
 axKHOpenAPI1.SetInputValue("기준일자", DateTime.Now.ToString("yyyyMMdd"));
 axKHOpenAPI1.SetInputValue("수정주가구분", "1");
 axKHOpenAPI1.CommRqData("주식일봉차트조회", "opt10081", 0,
"8100");
 }
 }
}

```

```

 }

 }

}

public void onEventConnect(object sender,
AxKHOpenAPILib._DKHOpenAPIEvents_OnEventConnectEvent e)
{
 if (e.nErrCode == 0)
 {

 stockList = new List<stockInfo>();

 string stocklist = axKHOpenAPI1.GetCodeListByMarket(null);
 string[] stockArray = stocklist.Split(';');
 AutoCompleteStringCollection stockCollection = new
AutoCompleteStringCollection();

 for (int i = 0; i < stockArray.Length; i++)
 {
 stockList.Add(new stockInfo(stockArray[i],
axKHOpenAPI1.GetMasterCodeName(stockArray[i])));

 stockCollection.Add(axKHOpenAPI1.GetMasterCodeName(stockArray[i]));
 }
 stockTextBox.AutoCompleteCustomSource = stockCollection;
 }
}

class stockInfo
{
 public string stockCode;
 public string stockName;
 public stockInfo(string stockCode, string stockName)
 {
 this.stockCode = stockCode;
 this.stockName = stockName;
 }
}

class priceInfoObject
{
 public string 일자;
}

```

```
public int 시가;
public int 고가;
public int 저가;
public int 종가;
public priceInfoObject(string 일자 , int 시가 , int 고가 , int 저가 , int 종가)
{
 this.일자 = 일자;
 this.시가 = 시가;
 this.고가 = 고가;
 this.저가 = 저가;
 this.종가 = 종가;

}

}
```

## 부록 B : 소스코드

### Preview

12장에서는 10장에서 만들어본 자동매매 프로그램의 소스코드를 담고 있습니다.

### Contents

12.1장에는 Github 주소를 작성했습니다. Github 주소에는 10장에서 만들어본 자동매매 프로그램의 소스코드가 작성되어 있습니다.

# Github

Github

<https://github.com/WooJongSeon/JSWoo-AutoStockTrading/blob/master/SourceCode%2CReadME>

The screenshot shows a GitHub repository page for 'JSWoo-AutoStockTrading'. The top navigation bar includes links for Features, Business, Explore, Marketplace, Pricing, and Sign in or Sign up. The repository header shows 'This repository' and a search bar. Below the header, there are buttons for Watch (0), Star (0), and Fork (0). The main navigation tabs are Code (selected), Issues (0), Pull requests (0), Projects (1), and Insights.

The repository path is 'WooJongSeon / JSWoo-AutoStockTrading' under the 'JSWoo-AutoStockTrading / SourceCode,ReadME' branch. A 'Find file' and 'Copy path' button are visible. The commit history shows a single commit by 'WooJongSeon' with the message 'Create SourceCode,ReadME' and timestamp 'b266b67 2 minutes ago'. There is 1 contributor listed.

The code editor displays 754 lines (668 sloc) and 39.8 KB. The code content is as follows:

```

1 using System;
2 using System.Collections.Generic;
3 using System.Windows.Forms;
4
5 namespace WindowsFormsApp12
6 {
7 public partial class Form1 : Form
8 {
9 List<StockInfo> stockList;
10 List<AutoTradingRule> autoTradingRuleList;
11 List<stockBalance> stockBalanceList;
12 List<outstanding> outstandinglist;
13
14 int autoScreenNumber = 1000;
15 int autoRuleID = 0;
16 string currentCondition="";
17 public static string ACCOUNT_NUMBER="";
18 }
19 }
```