iDeal'den Lib.cs/User.dll Kütüphanelerinize Veri Aktarım Yöntemleri

Öncelikle amatör olduğum için isimleri filan yanlış kullanmış olabilirim. Kusura bakmayın.
Kolaylık olsun diye Lib.cs üzerinde göstereceğim, aynı şeyleri User.dll uygulayabilirisiniz.

Property/Özellik Yöntemi
C#'ta yazılan programın özelliklerini saklamak için kullanılan bir yöntemdir. Koca koca listeleri bu property'lerde saklamak ne kadar doğru bilmiyorum ama NinjaTrader'dan böyle gördük.
Lib.cs:

```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {
   public float Carpan { get; set; }
   public List<float> Kapanis { get; set; }
   public List<float> Sonuc { get; set; }

   public void Hesapla()
   {
    Sonuc = new List<float>(new float[Kapanis.Count]);
    for(int i=0;i<Kapanis.Count;i++)
     Sonuc[i] = Kapanis[i] * Carpan;
   }
 }
}
//***kod***
```

Kütüphanemizin içine float tipinde 3 değişken tanımladık. Bunlardan iki tanesi dikkat ederseniz liste olarak tanımlı. Kapanis isimli değişken liste iDeal'den gelen kapanış verilerini saklayacak, Carpan isimli değişken yine iDeal'den gelen çarpan değerini saklayacak, Sonuc isimli değişken liste ise Hesapla metodunda yapılan çarpım işleminin sonucunu saklayacak. Burada dikkat edilecek diğer bir nokta, Hesapla metodunun içerisinde Sonuc = new List<float>(new float[Kapanis.Count]); şeklinde iDeal'den gelen kapanış listesinin büyüklüğünde yeni bir liste oluşturuyoruz. Daha sonra for döngüsü ile listenin her bir elemanını hesaplatıp Sonuc değişkenine yazdırıyoruz. Hesaplanan veri artık listede saklı. Şimdi iDeal'den hesaplanan veriyi okutup ekrana çizdireceğiz.
iDeal:

```
//***kod***
Lib.Kapanis = Sistem.GrafikFiyatSec("Kapanis"); //Kapanış fiyatını doğrudan kütüphanedeki Kapanis değişkenine atadık.
Lib.Carpan = 1.1f; //Kütüphanedeki Carpan değişkenine 1.1float değerini atadık.
Lib.Hesapla(); //Hesapla metodunu çağırarak çarpma işlemini yaptırdık.
Sistem.Cizgiler[0].Deger = Lib.Sonuc; //Kütüphanemizdeki Sonuc değişkenini ekrana çizdirdik.
//***kod***
```

Yukarıdaki kodu iDeal içerisine yazıp sistemi grafik üzerine atarsanız, kapanış fiyatının 1.1 ile çarpımının sonucunu ekrana çizdirirsiniz. Basit bir örnek gösterdim, bunu daha pek çok şekilde kullanabilirsiniz.

Method/Metot Yöntemi

Property yönteminde gördüğünüz gibi, property'ler sadece veri saklamaya yarıyor(şimdilik). Herhangi bir hesaplama yapmak istediğimizde bir metot yazmak zorunda kalıyoruz. Bu yöntemde veriyi direk metoda gönderip geriye almayı göstereceğiz.

Lib.cs:

```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {

  public List<float> Hesapla(List<float> Kapanis, float Carpan)
  {
   var Sonuc = new List<float>(new float[Kapanis.Count]);
   for (int i = 0; i < Kapanis.Count; i++)
    Sonuc[i] = Kapanis[i] * Carpan;
   return Sonuc;
  }
 }
}
//***kod***
```

Property örneğindeki Hesapla() metodu ile bu örnekteki Hesapla() metodunu karşılaştıralım.

İlk örnekte public void Hesapla() yani burdaki void terimi geriye veri döndürmeyen metotlar için kullanılır. Geriye veri döndürmedik ama metot içerisinde Sonuc property'sine yazarak veriyi almıştık.

Bu örnekte ise void yerine List<float> diyerek geriye float veri tipinde bir liste döndüreceğimizi tanımlamış olduk.

Hesapla(List<float> Kapanis, float Carpan) Burada da Hesapla metodunun alabileceği parametreleri belirledik.

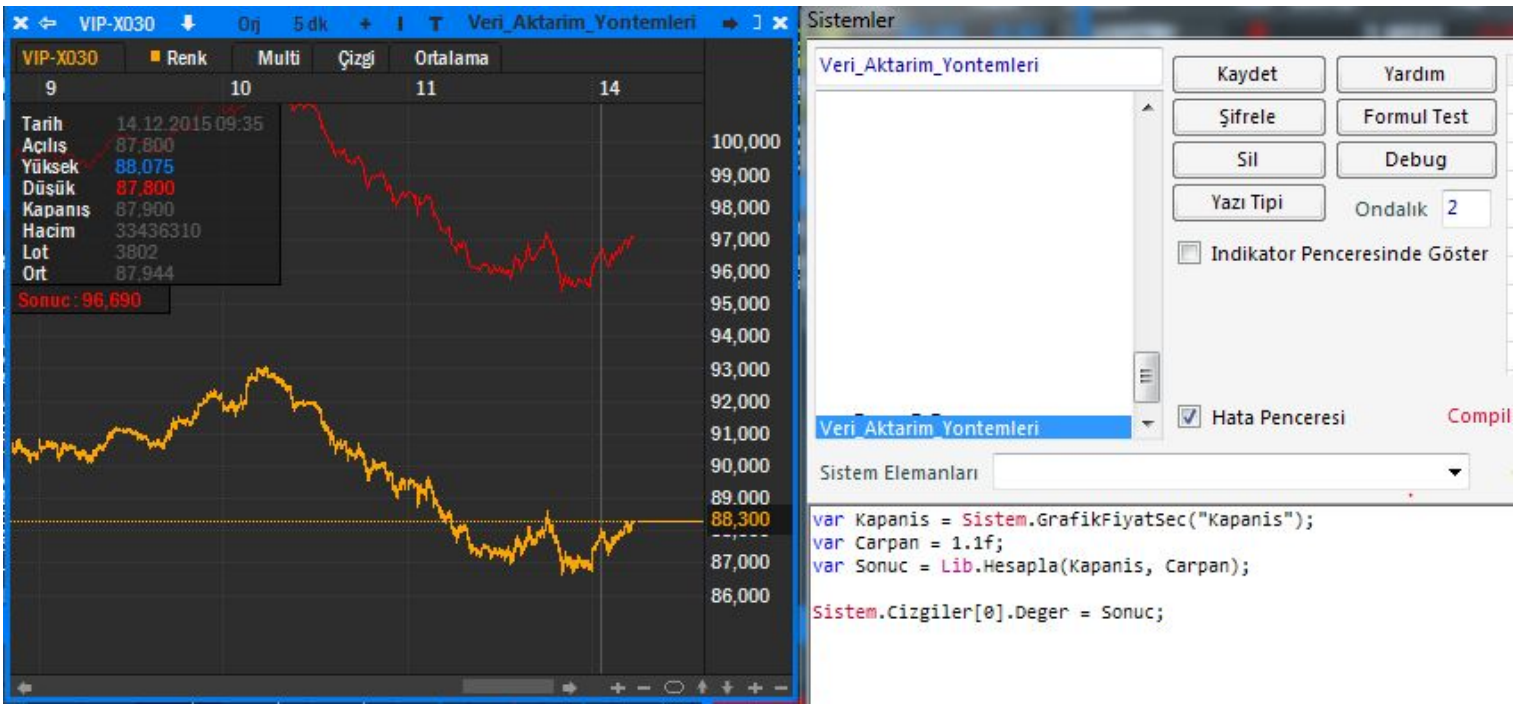
Önceki örnekte property olarak tanımladığımız Kapanis ve Carpan değişkenlerini burada parametre olarak tanımladık. Böylece Kapanis ve Carpan verilerini doğruca metodun içinde kullanılacak şekilde metoda göndereceğiz.

Metodun içine girecek olursak Sonuc adında Kapanis listesi ile eşit uzunlukta yeni boş bir liste tanımlayıp, daha sonra for döngüsü içerisinde hesaplamayı yaptırıp bu boş listeyi dolduruyoruz. En son satırda return Sonuc; diyerek bu hesaplanmış listeyi geriye döndürüyoruz.

iDeal:

```
//***kod***
var Kapanis = Sistem.GrafikFiyatSec("Kapanis"); //Kapanis adında bir değişken oluşturup kapanış verilerini atıyoruz.
var Carpan = 1.1f; //Carpan değişkenini tanımlayıp değer atıyoruz.
var Sonuc = Lib.Hesapla(Kapanis, Carpan); //Burası uzun olduğu için aşağıda anlatıcaz.
Sistem.Cizgiler[0].Deger = Sonuc; //Sonuc değişkenini çizdiriyoruz.
//***kod***
```

Yukarıdaki kodda göreceğiniz üzere, Kapanis, Carpan ve Sonuc diye 3 değişken tanımladık.
Kapanis değişkenine iDeal'den aldığımız kapanış fiyatlarını, Carpan değişkenine kendi atadığımız değeri yazdık. Sonuc değişkenine ise kütüphanemizde tanımladığımız metodu ve parantez içerisine bu metodun alabileceği parametreler olan Kapanis listesi ile Carpan değerini tanımladık. Böylece sistem bu parametreleri alıp Hesapla() metoduna gitti, çarpım işlemini yapıp listeyi doldurdu ve bu listeyi Sonuc değişkenine geri gönderdi. Bize geriye kalan ekrana çizdirmek.



Şimdi iki yöntemin birlikte kullanıldığı biraz karışık bir örnek yazalım.
Lib.cs:

```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {
  public List<float> Toplam { get; set; }

  public void Topla(List<float> Yuksek, List<float> Dusuk)
  {
   Toplam = new List<float>(new float[Yuksek.Count]);
   for (int i = 0; i < Yuksek.Count; i++)
    Toplam[i] = Yuksek[i] + Dusuk[i];
  }

  public List<float> Bol(float Bolen)
  {
   var Sonuc = new List<float>(new float[Toplam.Count]);
   for (int i = 0; i < Toplam.Count; i++)
    Sonuc[i] = Toplam[i] / Bolen;
```

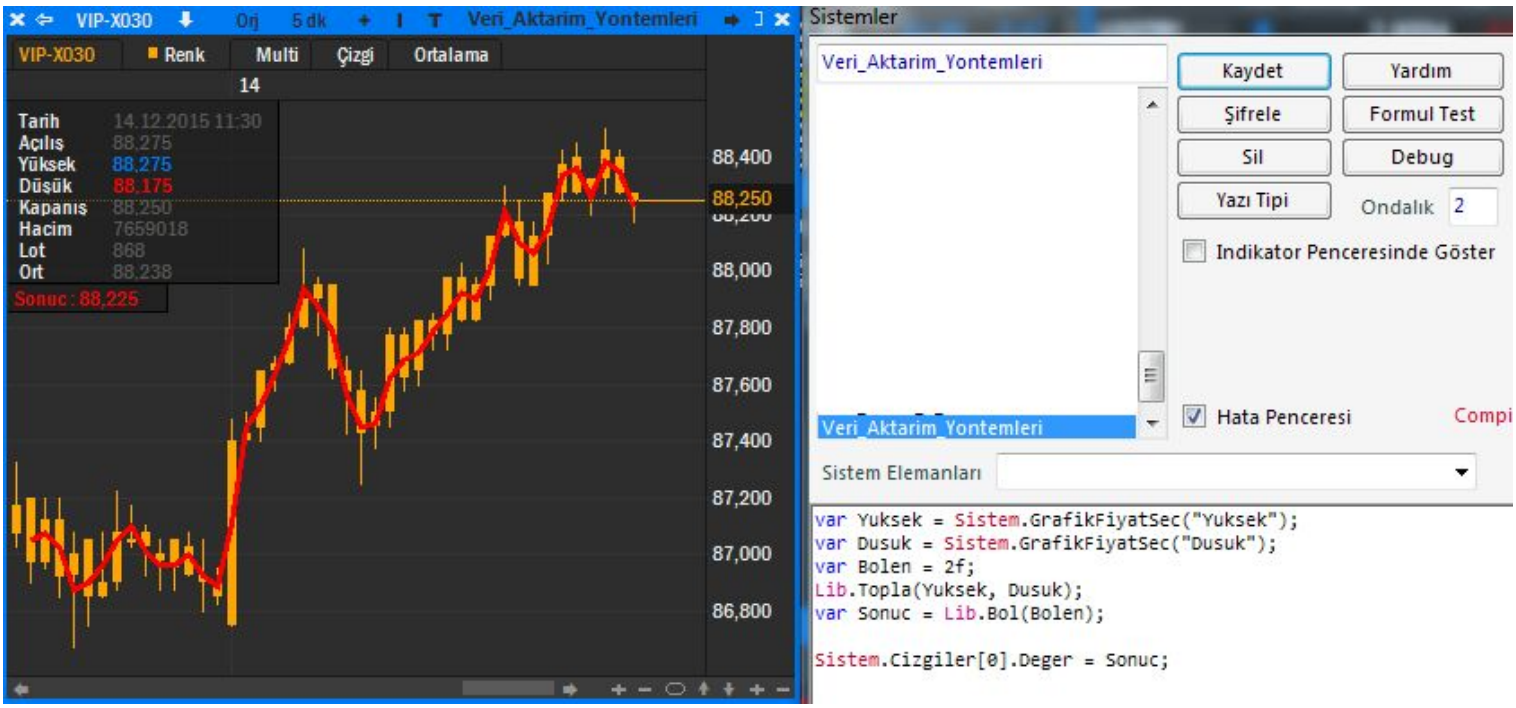
```
  return Sonuc;
 }
}
}
```

//*kod*****

Toplam adında float veri tipinde liste olacak şekilde bir property tanımladık.
void türünde yani geriye veri döndürmeyen, 2 tane float listesini parametre olarak alacak şekilde Topla adında bir metot tanımladık. Bu metot, göndereceğimiz yüksek düşük verilerini toplayıp Toplam property'sine yazacak.
Daha sonra geriye float list tipinde veri gönderen 1 tane float türünden parametresi olan Bol adında bir metot tanımladık. Bu metot da Toplam property'sinde yazılı olan veriyi alıp bizim gönderdiğimiz Bolen değerine bölerek oluşan yeni listeyi geriye döndürecek.
iDeal:

```
//***kod***
var Yuksek = Sistem.GrafikFiyatSec("Yuksek"); //Yuksek değişkenine yüksek verilerini atadık
var Dusuk = Sistem.GrafikFiyatSec("Dusuk"); //Dusuk değişkenine düşük verilerini atadık
var Bolen = 2f; //Bolen değişkenine 2float değerini atadık
Lib.Topla(Yuksek, Dusuk); //Yuksek,Dusuk parametreleri ile Topla metodunu çağırdık
var Sonuc = Lib.Bol(Bolen); //Bolen parametresi ile Bol metodunu çağırdık ve dönen listeyi Sonuc değişkenine atadık
Sistem.Cizgiler[0].Deger = Sonuc; //Çizdirdik
//***kod***
```

Burada dikkat edilecek nokta geri veri döndürmeyen Topla metodu ile geri veri döndüren Bol metodunun nasıl kullanıldığını iyi inceleyin. Bu örneği diğer örneklerle karşılaştırarak aradaki farkları çalışma şeklini filan, daha iyi algılayabilirsiniz. Özet olarak H+L/2 yani, yüksek ile düşüğün toplamının 2 ye bölünmesi ile iDeal'de tanımlı olan OrtaNokta verisini kendi kütüphanemizde hesaplamış olduk.



Şimdi biraz daha karışık yöntemlere girelim.
İşleriniz arttıkça daha çok veriye ihtiyacınız olacak ve bir süre sonra aşağıdaki gibi bir metot parametreleri yazmak zorunda kalabilirsiniz.

public void Hesapla(List<DateTime> Tarih, List<float> Kapanis, List<float> Yuksek, List<float> Dusuk, List<float> Acilis, List<float> Hacim, float Deger1, int Deger2, bool Deger3)

Uzayıp gider.
iDeal, bir barın içindeki tüm bilgileri bir sınıfta saklar. Bunu cxBar sınıfı olarak tanımlamışlardır. Yukarıdaki gibi bar verilerini tek tek yazmak yerine komple cxBar sınıfını metoda gönderebilirsiniz.
Ama bunun için cxBar sınıfının kütüphanenizde tanımlı olması lazım ki bunu tanımlayamazsınız. Yani tam anlamıyla tanımlayamazsınız, yarım yamalak bir tanımlamada işe yaramayacaktır. Bunu aşmanın yolu Tiberius üstattan öğrendiğimiz dynamic tipidir. Bu tip derleme sırasında kontrol yapmaz, çalışma sırasında verileri yorumlayarak görevini yerine getirir. Tam olarak anlatamamış olabilirim, detay için google'a bakın. Bir örnek yazalım.

Lib.cs:

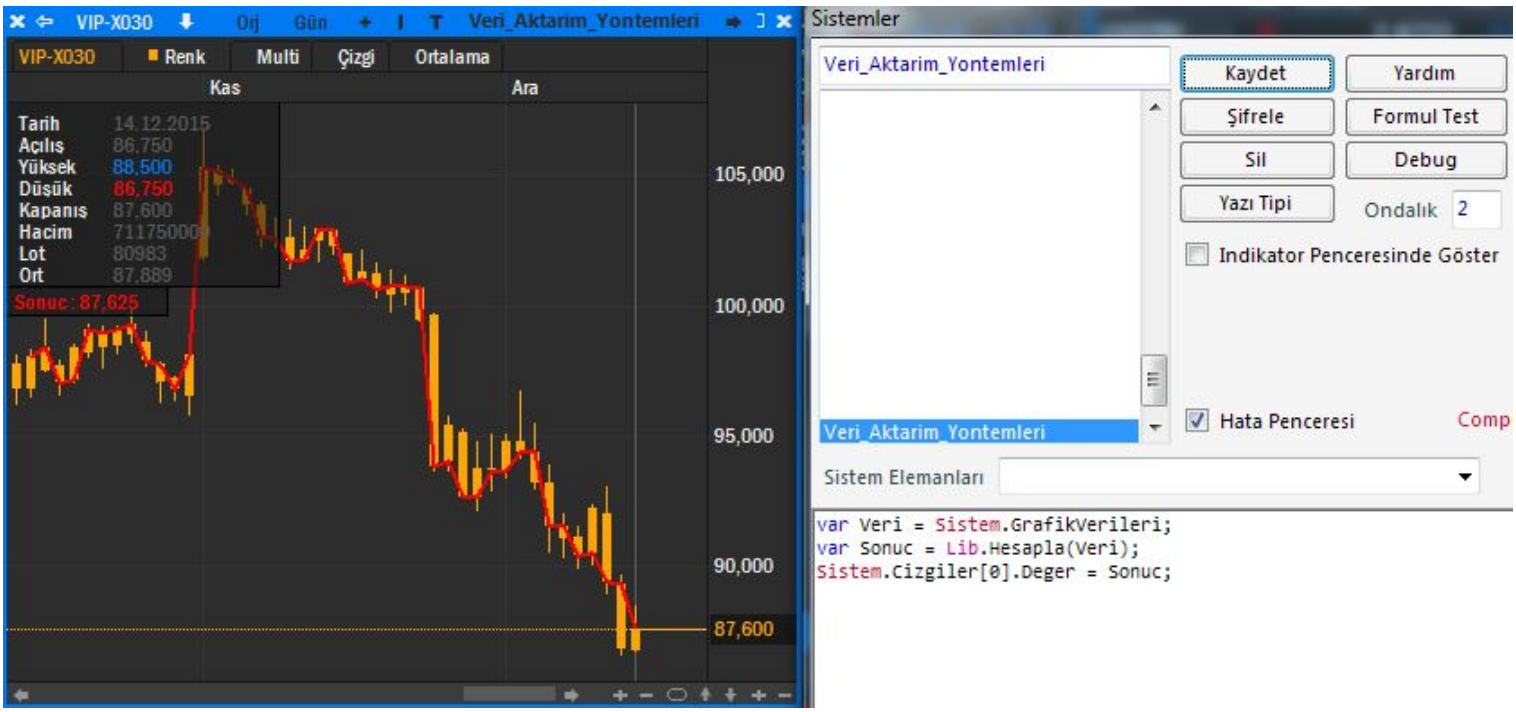
```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {

  public List<float> Hesapla(dynamic Veriler)
  {
   var liste = new List<float>(new float[Veriler.Count]);
   for (int i = 0; i < Veriler.Count; i++)
   {
    if (Veriler[i].Date.Day % 2 == 0) liste[i] = Veriler[i].Close;
    else liste[i] = Veriler[i].Open;
   }
   return liste;
  }
 }
}
//***kod***
```

Burada Hesapla metodu parametresine bakacak olursanız sadece dynamic Veriler tanımlıdır. Tek tek tüm veri listelerini göndermek yerine Veriler adındaki cxBar sınıfını dynamic tipinde göndererek hem kod hamallığından hemde kalabalığından kurtulmuş olduk.

iDeal:

```
//***kod***
var Veri = Sistem.GrafikVerileri; //GrafikVerileri ile tüm cxBar sınıfını Veri değişkenine atadık
var Sonuc = Lib.Hesapla(Veri); //Veri değişkeni ile birlikte Hesapla metodunu çağırdık ve dönen listeyi Sonuc'a aktardık
Sistem.Cizgiler[0].Deger = Sonuc; //Çizdirdik
//***kod***
```

Bu örnekte tek sayılı günlerde açılış, çift sayılı günlerde kapanış değerini alan bir liste oluşturup ekrana çizdirdik.



Gördüğünüz gibi daha basit bir yöntemmiş gibi gözükmekte. Dikkat edilmesi gereken nokta dynamic ile kullandığımız verinin(örnekte cxBar sınıfı) içeriğini çok iyi bilmeniz gerekmektedir. iDeal tarafında pek sıkıntı olmaz ama kütüphanenizde kod yazarken sınıfın içeriğine hakim değilseniz sorun yaşarsınız.

Peki Veriler adında tanımladığımız cxBar sınıfının içeriğini nereden bildik veya ileride başka sınıfların içeriğini nereden bileceğiz?
cxBar sınıfının içeriğini iDeal Profesyonelleri daha önceden paylaşmıştı.

```
//***kod***
public class cxBar
 {
 public DateTime Date;
 public float Open = 0;
 public float High = 0;
 public float Low = 0;
 public float Close = 0;
 public float Size = 0;
 public float Vol = 0;
 public float Opint = 0;
 public float AveragePrice()
 public float TypicalPrice()

 public static void Read(string symbolX, string periodX, List<cxBar> dataListX)
 public static List<float> PeriodAdjust(List<cxBar> lowbarsX, List<cxBar> highbarsX, List<float> dataX)
 }
//***kod***
```

Bu paylaşım sayesinde Veriler isimli cxBar sınıfının içeriğine hakim olabiliyoruz.
İçeriğini bilmediğimiz bir sınıfı dynamic tipinde kullanabilmek için debug modunda breakpoint ile durdurup izin verilen elemanları görebilirsiniz.

Class/Sınıf Yöntemi

cxBar sınıfı gibi, kütüphanemizde kendi sınıfımızı oluşturabiliriz. iDeal'in cxBar sınıfına benzer bir örnek yapalım.
Lib.cs:

```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {
  public List<cxBar> VeriListesi { get; set; }

  public void VeriAktar(dynamic Veriler)
  {
   VeriListesi = new List<cxBar>();
   for(int i=0;i<Veriler.Count;i++)
   {
    VeriListesi.Add(new cxBar
    {
     Date = Veriler[i].Date,
     Open = Veriler[i].Open,
     High = Veriler[i].High,
     Low = Veriler[i].Low,
     Close = Veriler[i].Close,
     Size = Veriler[i].Size,
     Vol = Veriler[i].Vol,
     Opint = Veriler[i].Opint
    });
   }
  }
```

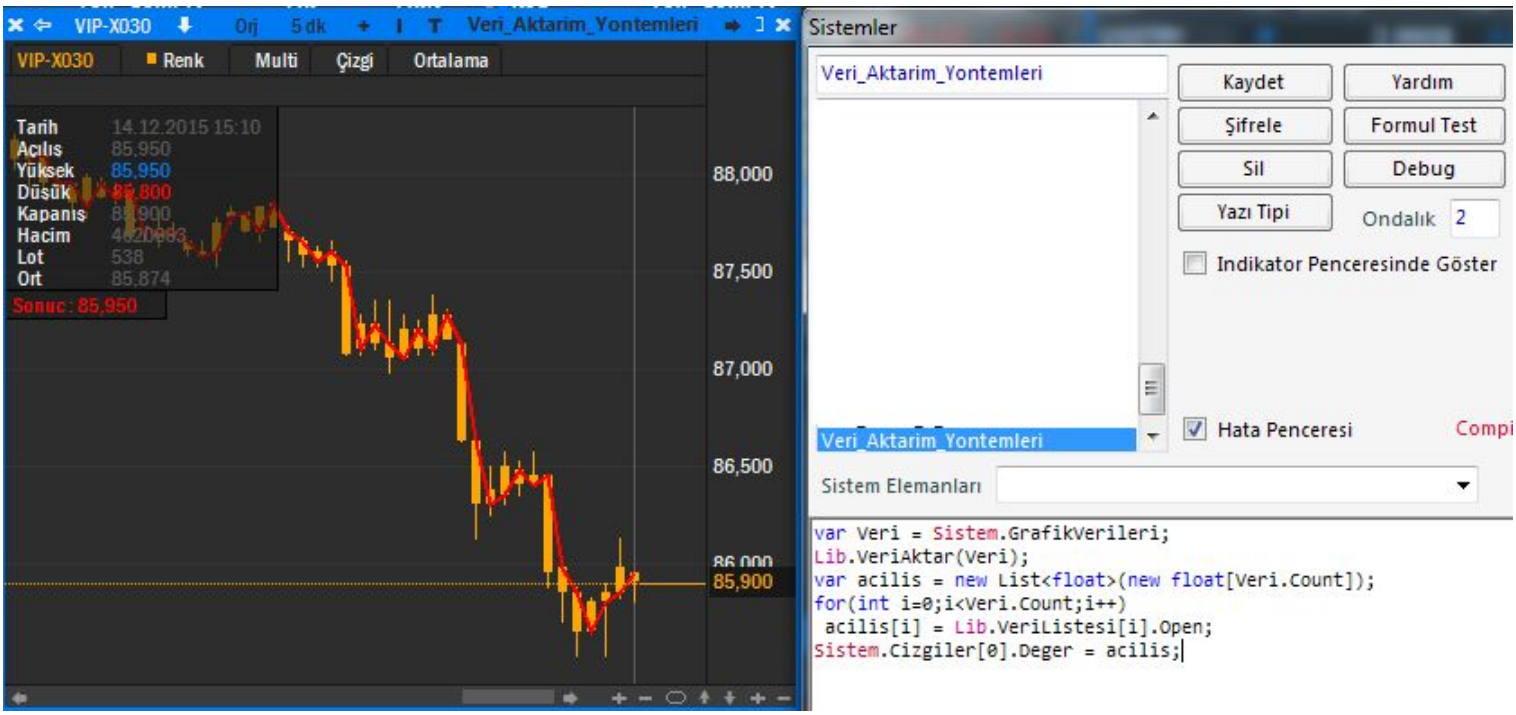
```
 public class cxBar
 {
  public DateTime Date;
  public float Open = 0;
  public float High = 0;
  public float Low = 0;
  public float Close = 0;
  public float Size = 0;
  public float Vol = 0;
  public float Opint = 0;
 }
}
}
//***kod***
```

Burada içerisinde Date,Open,High,Low,Close,Size,Vol,Opint isimli değişkenler olan bir cxBar sınıfı yazdık. Daha sonra VeriListesi adında bir property tanımladık. Bu property'nin cxBar sınıfından oluşan bir liste olduğuna dikkat edin. VeriAktar adında bir metot ile iDeal'den dynamic tipinde gelen veriyi alıp kütüphanemizdeki cxBar sınıfını kullanan listeye yazdırdık.

iDeal:

```
//***kod***
var Veri = Sistem.GrafikVerileri;
Lib.VeriAktar(Veri);
var acilis = new List<float>(new float[Veri.Count]);
for(int i=0;i<Veri.Count;i++)
 acilis[i] = Lib.VeriListesi[i].Open;
Sistem.Cizgiler[0].Deger = acilis;
//***kod***
```

iDeal tarafına bakacak olursak sırası ile, GrafikVerilerini Veri değişkenine atadık, Veri değişkenini VeriAktar metodu ile gönderdik, acilis isimli bir değişken liste oluşturduk, for döngüsü ile kendi sınıfımızdaki açılış fiyatını acilis değişkenine atadık ve çizdirdik. Bunu niye yaptık diye düşünmeyin örnek olsun diye yaptık işte.



Şimdi biraz daha karışık işlere bakalım, iDeal'den sadece grafik verilerini değil iDeal Profesyonellerinin izin verdiği herşeyi kendi kütüphanenize çekebilirsiniz.

iDeal:

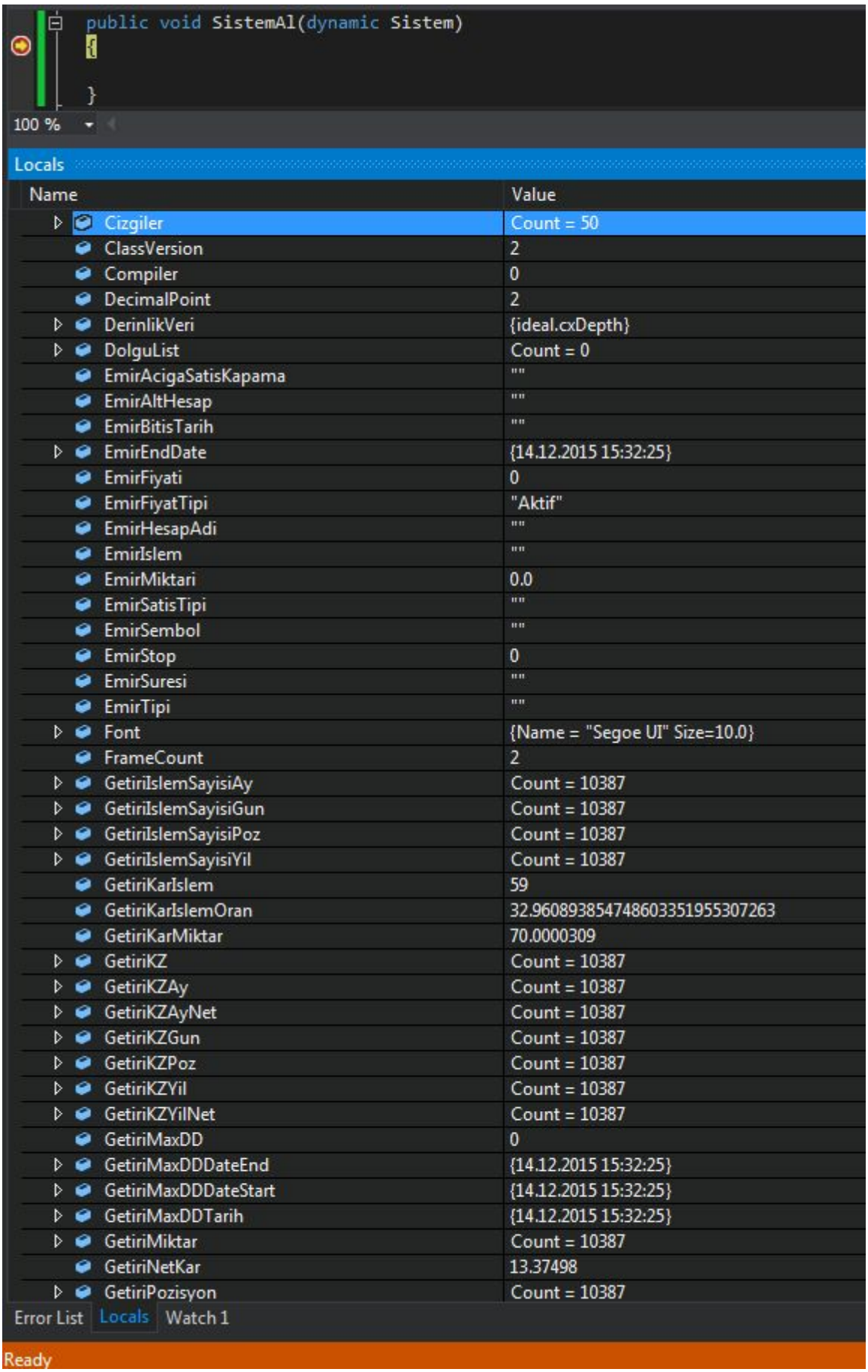
```
//***kod***
var veri = Sistem.GrafikVerileri;
var C = Sistem.GrafikFiyatSec("Kapanis");
var MA1 = Sistem.MA(C, "Exp", 20);
```

```
var MA2 = Sistem.MA(C, "Exp", 50);
var sonyon = "";

for(int i=1;i<veri.Count-1;i++)
{
if(MA1[i]>MA2[i] && MA1[i-1]<MA2[i-1] && sonyon != "A")
{
Sistem.Yon[i] = "A"; sonyon="A";
}
else if(MA1[i]<MA2[i] && MA1[i-1]>MA2[i-1] && sonyon != "S")
{
Sistem.Yon[i] = "S"; sonyon="S";
}
}

Sistem.GetiriHesapla("01/01/2015",0);
var KZ = Sistem.GetiriKZ;

Sistem.Cizgiler[0].Deger = MA1;
Sistem.Cizgiler[1].Deger = MA2;
Sistem.Cizgiler[2].Deger = KZ;
//***kod***
```

Örnek olarak klasik ema kesiştiren bir sistem yazıyoruz. Bunu Veri_Aktarim_Demo olarak kaydediyoruz.
Daha sonra başka bir sistem içerisine:

```
//***kod***
User.SistemAl(Sistem.SistemGetir("Veri_Aktarim_Demo","VIP'VIP-X030","5"));
//***kod***
```

Yazarak demo sistemi çağırıp, iDeal'in tüm Sistem sınıfını olduğu gibi User.SistemAl metoduna dynamic tipinde gönderiyoruz.
Debug modunda breakpoint ile durdurup Sistem sınıfının içerisindeki izin verilen elemanları görebiliyoruz.



Liste bu şekilde uzayıp gidiyor, meraklısı girip bakabilir. Bu Sistem sınıfını kendi kütüphanenize göndererek birçok kullanışlı veriye erişebilirsiniz.

Son olarak tamamen kendi kütüphanenizde sistem nasıl yazılır buna bir örnek yazalım.
Öncelikle basit bir algoritma yazalım. (Basit diyorum ama algoritma hazırlamak çok önemli google'da araştırın)
iDeal'den veriyi dynamic tipinde çekeriz
Alınan veri ile ema listelerini hesaplatılırız
Hesaplatılan ema listeleri ile alsat sistemimizi oluştururuz.

Sistemi iDeal'e gönderip grafiğe çizdiririz.
Bu algoritmaya göre ema'nın formülüne ihtiyacımız var, zaten ideal formülleri sayfasında eskiden yayınlamıştık, oradan kopyalıyoruz.
Lib.cs:

```
//***kod***
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ideal
{
 public class Lib
 {
  public List<float> EMA1 { get; set; }
  public List<float> EMA2 { get; set; }
  public List<string> SonIslem { get; set; }
  public List<float> SonIslemFiyati { get; set; }
  public List<float> KarZarar { get; set; }
  public List<float> MaksKZ { get; set; }
  public List<float> Puan { get; set; }
  public List<float> AnlikPuan { get; set; }
  public List<float> KomEgrisi { get; set; }
  public List<float> KaymaEgrisi { get; set; }
  public List<float> DD { get; set; }
  public List<float> MDD { get; set; }
  public List<float> KarZararIs { get; set; }
  public List<float> KarZararAy { get; set; }

  public void Grafik(dynamic Veriler)
  {
   if (Veriler.Count != 0)
   {
    int Kontrat = 10;
    float KomOran = 0.00015f;
    float Kayma = 12.5f;
    EMA1 = EMA(20, Veriler);
    EMA2 = EMA(50, Veriler);
    KomEgrisi = new List<float>(new float[Veriler.Count]);
    KaymaEgrisi = new List<float>(new float[Veriler.Count]);
    var IslemMiktari = new List<float>(new float[Veriler.Count]);
    var Pozisyon = new List<float>(new float[Veriler.Count]);
    KarZarar = new List<float>(new float[Veriler.Count]);
    Puan = new List<float>(new float[Veriler.Count]);
    AnlikPuan = new List<float>(new float[Veriler.Count]);
    var Komisyon = new List<float>(new float[Veriler.Count]);
    SonIslem = new List<string>(new string[Veriler.Count]);
    SonIslemFiyati = new List<float>(new float[Veriler.Count]);
    MaksKZ = new List<float>(new float[Veriler.Count]);
    DD = new List<float>(new float[Veriler.Count]);
    MDD = new List<float>(new float[Veriler.Count]);
    var SonPozisyon = 0f;
    float ToplamKarZarar = 0f;
    float ToplamKomi = 0f;
    int Kantir = 0;
    int IslemBar = 1;

    for (int i = 1; i < Veriler.Count; i++)
    {
```

```
    if (EMA1[i] > EMA2[i] && EMA1[i - 1] < EMA2[i - 1] && SonPozisyon <= 0)
    {
     if (SonPozisyon == 0) IslemMiktari[i] = Kontrat;
     else if (SonPozisyon < 0) IslemMiktari[i] = Kontrat * 2;
     SonIslemFiyati[i] = Veriler[i].Close;
     SonIslem[i] = "AL";
     Kantir = 1;
     IslemBar = i;
     SonPozisyon = SonPozisyon + IslemMiktari[i];
     ToplamKarZarar = ToplamKarZarar - SonIslemFiyati[i] * IslemMiktari[i];
     ToplamKomi = SonIslemFiyati[i] * KomOran * IslemMiktari[i];
     KomEgrisi[i] = KomEgrisi[i - 1] + Math.Abs(ToplamKomi) * 100;
     KaymaEgrisi[i] = KaymaEgrisi[i - 1] + Math.Abs(Kayma * IslemMiktari[i] / 10);
    }
    else if (EMA1[i] < EMA2[i] && EMA1[i - 1] > EMA2[i - 1] && SonPozisyon >= 0)
    {
     if (SonPozisyon == 0) IslemMiktari[i] = Kontrat * -1;
     else if (SonPozisyon > 0) IslemMiktari[i] = Kontrat * -2;
     SonIslemFiyati[i] = Veriler[i].Close;
     SonIslem[i] = "SAT";
     Kantir = 1;
     IslemBar = i;
     SonPozisyon = SonPozisyon + IslemMiktari[i];
     ToplamKarZarar = ToplamKarZarar - SonIslemFiyati[i] * IslemMiktari[i];
     ToplamKomi = SonIslemFiyati[i] * KomOran * IslemMiktari[i];
     KomEgrisi[i] = KomEgrisi[i - 1] + Math.Abs(ToplamKomi) * 100;
     KaymaEgrisi[i] = KaymaEgrisi[i - 1] + Math.Abs(Kayma * IslemMiktari[i] / 10);
    }
    else { SonIslem[i] = SonIslem[i - 1]; KomEgrisi[i] = KomEgrisi[i - 1]; KaymaEgrisi[i] = KaymaEgrisi[i - 1];
SonIslemFiyati[i] = SonIslemFiyati[i - 1]; }
    Pozisyon[i] = SonPozisyon;
    if (Kantir == 1) { AnlikPuan[i] = (ToplamKarZarar + SonIslemFiyati[i] * Pozisyon[i]) * 100; Kantir = 0; }
    else AnlikPuan[i] = (ToplamKarZarar + Veriler[i].Close * Pozisyon[i]) * 100;
    Puan[i] = (ToplamKarZarar + SonIslemFiyati[i] * Pozisyon[i]) * 100;
    Komisyon[i] = Math.Abs(ToplamKomi) * 100;
    KarZarar[i] = Puan[i] - KaymaEgrisi[i] - KomEgrisi[i];
    MaksKZ[i] = KarZarar[i] <= MaksKZ[i - 1] ? MaksKZ[i - 1] : KarZarar[i];
    DD[i] = KarZarar[i] < MaksKZ[i] ? KarZarar[i] - MaksKZ[i] : 0f;
    MDD[i] = DD[i] >= MDD[i - 1] ? MDD[i - 1] : DD[i];
   }
   KarZararAy = new List<float>(new float[Veriler.Count]);
   float KarZararAy1 = 0;
   int BarNoAy1 = 1;
   float FarkAy = 0;
   KarZararIs = new List<float>(new float[Veriler.Count]);
   float KarZararIs1 = 0;
   int BarNoIs1 = 1;
   float FarkIs = 0;
   for (int i = 1; i < Veriler.Count; i++)
   {
    if (Veriler[i].Date.Month != Veriler[i - 1].Date.Month)
    {
     FarkAy = KarZarar[i] - KarZararAy1;
     for (int j = BarNoAy1; j < i; j++)
      KarZararAy[j] = FarkAy;
     KarZararAy1 = KarZarar[i];
     BarNoAy1 = i;
    }
    if (SonIslem[i] == "AL" && SonIslem[i - 1] != "AL" || SonIslem[i] == "SAT" && SonIslem[i - 1] != "SAT")
    {
```

```
     FarkIs = KarZarar[i] - KarZararIs1;
     for (int j = BarNoIs1; j < i; j++)
      KarZararIs[j] = FarkIs;
     KarZararIs1 = KarZarar[i];
     BarNoIs1 = i;
    }
    if (i == Veriler.Count - 2)
    {
     FarkAy = KarZarar[i] - KarZararAy1;
     for (int j = BarNoAy1; j < Veriler.Count; j++)
      KarZararAy[j] = FarkAy;
     FarkIs = KarZarar[i] - KarZararIs1;
     for (int j = BarNoIs1; j < Veriler.Count; j++)
      KarZararIs[j] = FarkIs;
    }
   }
  }
 }

 private static List<float> EMA(int Periyot, dynamic Veriler)
 {
  var liste = new List<float>(new float[Veriler.Count]);
  if (Veriler.Count > Periyot)
  {
   liste[0] = Veriler[0].Close;
   float carpan = 2f / (Periyot + 1f);
   for (int i = 1; i < Veriler.Count; i++)
   {
    if (i >= Periyot)
     liste[i] = liste[i - 1] + carpan * (Veriler[i].Close - liste[i - 1]);
    else if (i > 0)
     for (int j = 1; j <= i; j++)
      liste[i] = liste[j - 1] + carpan * (Veriler[j].Close - liste[j - 1]);
   }
  }
  return liste;
 }
}
}
//***kod***
```

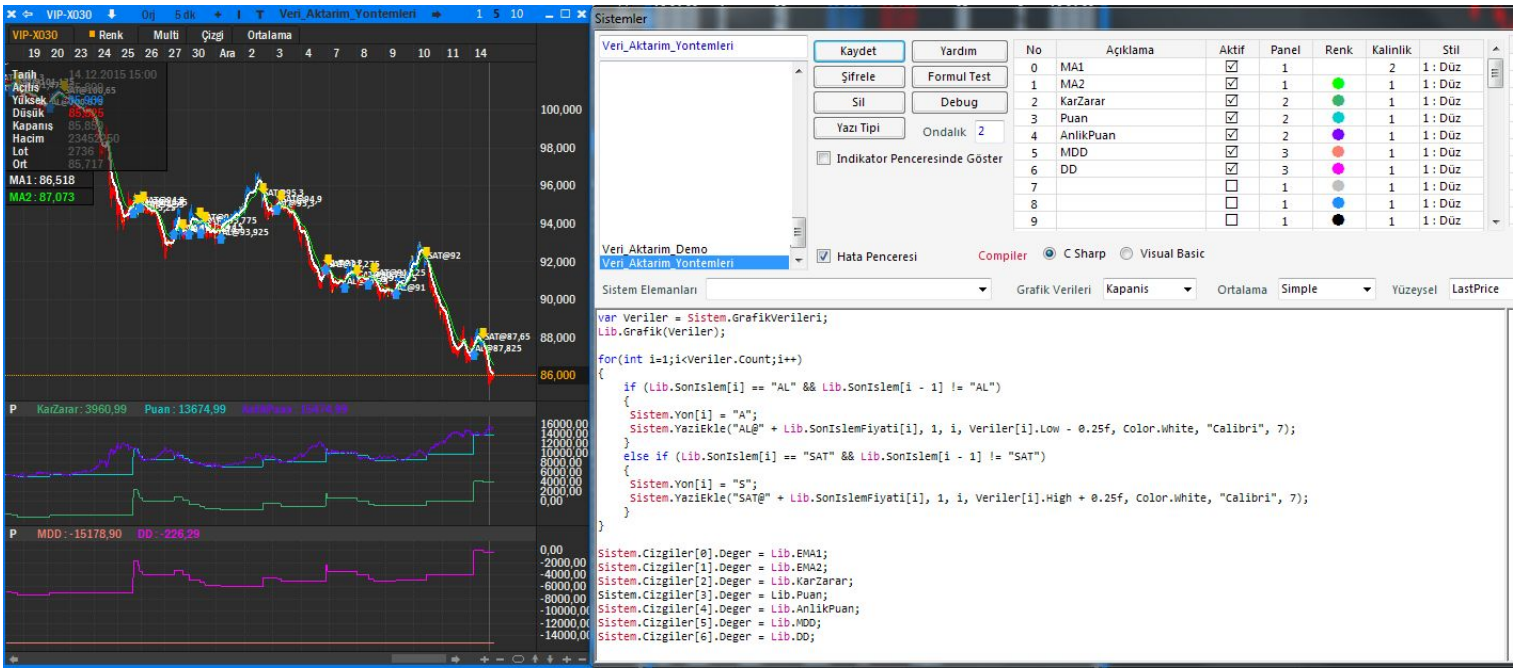
Lib.cs içerisine sistemimizi yazıyoruz. En üstte grafiğe çizdirmek için kullanacağımız listeleri property olarak tanımladık, daha sonra Grafik metodu ile ema'ları hesaplatıp alsat koşulumuzuda tanımlayarak sistemi oluşturduk. Grafik metodu geriye veri döndürmüyor ama hesapladığı listeleri tanımladığımız property'lere yazıyor. Böylece iDeal üzerinden property'lere erişiyoruz. En alttada EMA'yı hesaplayan metot var. Grafik metodu içerisinde EMA metoduna erişip geri dönen listeleri alıp sistemde kullanıyor. Daha fazla detaya girmeyeceğim.

iDeal:

```
//***kod***
var Veriler = Sistem.GrafikVerileri;
Lib.Grafik(Veriler);

for(int i=1;i<Veriler.Count;i++)
{
   if (Lib.SonIslem[i] == "AL" && Lib.SonIslem[i - 1] != "AL")
   {
    Sistem.Yon[i] = "A";
    Sistem.YaziEkle("AL@" + Lib.SonIslemFiyati[i], 1, i, Veriler[i].Low - 0.25f, Color.White, "Calibri", 7);
   }
   else if (Lib.SonIslem[i] == "SAT" && Lib.SonIslem[i - 1] != "SAT")
   {
```

```
    Sistem.Yon[i] = "S";
    Sistem.YaziEkle("SAT@" + Lib.SonIslemFiyati[i], 1, i, Veriler[i].High + 0.25f, Color.White, "Calibri", 7);
   }
}

Sistem.Cizgiler[0].Deger = Lib.EMA1;
Sistem.Cizgiler[1].Deger = Lib.EMA2;
Sistem.Cizgiler[2].Deger = Lib.KarZarar;
Sistem.Cizgiler[3].Deger = Lib.Puan;
Sistem.Cizgiler[4].Deger = Lib.AnlikPuan;
Sistem.Cizgiler[5].Deger = Lib.MDD;
Sistem.Cizgiler[6].Deger = Lib.DD;
//***kod***
```

iDeal tarafında ise Verileri okuttuk ve Grafik metodunu çağırdık. Veriler gitti, hesaplandı ve geri döndü. Burda bir for döngüsü ile grafik üzerinde al-sat okları oluşturmak için sistemin al-sat sinyallerini tutan Lib.SonIslem listesi ile iDeal'in Sistem.Yon listesini eşleştirdik. Daha sonra hesaplanmış listeleri Sistem.Cizgiler sınıfına tanımladık ve ekrana çizdirdik.



Bu kadar, basitmiş dimi ;)

Evet şimdilik aklıma gelen yöntemler bunlar. Yukarıda belirttiğim gibi tekrar belirtmek istiyorum, amatör olduğum için yanlış terim kullanmış veya eksik yarım anlatmış olabilirim, idare edin. Eksik, fazla, hatalı bir yer olduğunu düşünüyorsanız veya anlamadığınız noktalar olmuşsa mesaj atabilirsiniz. İleriye doğru yeni şeyler geldikçe buraya eklemeye devam edebilirim belki.
Saygılar Ex