ENGİNE KÜTÜPHANESİ NEDİR?

Engine Kütüphanesi, algoritmalarınızı kolayca kodlayabilmeniz için bazı finansal indikatör ve fonksiyonları barındıran kütüphanedir. Engine kütüphanesi C# yazılım dili altyapısıyla oluşturulmuştur. Engine kütüphanesi içinde bulunan tüm metotları temel C# bilgisiyle kullanabilirsiniz. Kodlama editöründe "Engine." komutu ile kullanabileceğiniz bu kütüphane ile ilgili bilgilere bu makalede yer vereceğiz.

STRATEJİLERİNİZİ KODLAMAYA BAŞLARKEN

```
*/ Parametreler */

*/ Parametreler */

public void Load()
{
}

public void PriceChanged(Tick t)
{
}

public void OrderStatusChanged(Order o)
{
}

public void DepthChanged(Depth d)
{
}
```

Load: Bu fonksiyon stratejiniz çalıştığında ilk çalıştırılan ve sadece bir kere çalışan fonksiyondur. Bu fonksiyonda fiyat verisine abone olma, derinlik verisine abone olma, EmailNotification, SmsNotification gibi bir kere çalıştırılacak tanımlamalar ve fonksiyonlar kullanılmalıdır.

PriceChanged: Fiyat verisine abone olunduğunda abone olunan sembol' e ait fiyatların anlık veri akışı gelmektedir. Bu fonksiyon her fiyat değişiminde çalışmaktadır. Fiyat bazlı bir algoritma yazılacak ise bu fonksiyon içerisine kodlama yapılmalıdır.

Fonksiyon içerisine gelen Tick modeli:

```
{
    string Symbol              :Sembol
    string Market              :Pazar Tipi(IMKBH,VIP)
    double Price               :Fiyat
    double Change              :Değişim
    double Ask                 :Alım Fiyatı
    double Bid                 :Satım Fiyatı
    DateTime Date              :Tarih
    double ChangePercentage    :Değişim Yüzde
```

double High :Yüksek
double Low :Düşük
double TradeQuantity :Miktar
string Direction :Yön
double RefPrice :Referans Fiyatı
double BalancePrice :Ort.Fiyat
double BalanceAmount :Ort.Miktar
string Buying :Alan Kurum
string Selling :Satan Kurum
}

OrderStatusChanged: Bu fonksiyona gönderilen, iyileştirilen veya iptal edilen emrin geri dönüşü gelmektedir. Stratejiyi sanal çalıştırma da, emir iletilmediği için herhangi bir geri dönüş alınamadığından bu fonksiyon kullanılamaz.
Fonksiyon içerisine gelen Order modeli:
{
string Id :Emir ID
DateTime Date :Tarih
Directions Direction :Yön
string Symbol :Sembol
double Lot :Lot
PriceTypes PriceType :Fiyat Tipi
double Price :Fiyat
string Comment :Açıklama
OrderStatus Status :Emir Durumu
string Channel :Kanal
}
DepthChanged: Derinlik verisine abone olunduğunda abone olunan sembol' e ait derinliğin anlık veri akışı gelmektedir. Bu fonksiyon her derinlik değişiminde çalışmaktadır. Derinlik bazlı bir algoritma yazılacak ise bu fonksiyon içerisine kodlama yapılmalıdır.
Fonksiyon içerisine gelen Depth modeli:
{
string Symbol :Sembol
string Market :Pazar Tipi
string Direction :Yön
int Row :Satır
int Quantity :Lot Miktarı
double Price :Fiyat
int OrderCount :Emir Miktarı
DateTime Date :Tarih
}

Engine kütüphanesi kendi içinde kolaylık sağlayan bazı genel tanımlamalara sahiptir. Bu tanımları kodunuzun içinde istediğiniz yerde kullanabilirsiniz.

enum **Directions**: ***SendOrder*** fonksiyonunda kullanılan parametredir. *BUY* ve *SELL* olmak üzere iki tanımı vardır. *BUY* seçildiğinde alım, *SELL* seçildiğinde satış emri iletir.

bool **EmailNotification**: Kod içerisinde ***Load*** fonksiyonu içerisinde kullanılmalıdır. Standart olarak **false** tanımlıdır. Gönderilen emrinizin geri dönüşünü **email** olarak iletilmesini istiyorsanız **true** olarak tanımlanmalıdır.

EmailNotification=true;

bool **SmsNotification**: Kod içerisinde ***Load*** fonksiyonu içerisinde kullanılmalıdır. Standart olarak **false** tanımlıdır. Gönderilen emrinizin geri dönüşünü **sms** olarak iletilmesini istiyorsanız **true** olarak tanımlanmalıdır.

SmsNotification=true;

enum **MessageTypes**: ***SendMessage*** fonksiyonunda kullanılır. Detaylı anlatımı fonksiyon içinde mevcuttur.

enum **MovingAverageMethods**: Moving Average indikatörünün bir parametresidir.

Simple: Basit
Exponential: Üssel
Weighted: Ağırlıklı
Wilder: Welles Wilder (Hesaplayan kişinin adı ile anılmaktadır.)
TimeSeries: Time Series Forecast
Triangular: Üçgensel
Variable: Değişken
VolumeAdjusted: Hacim Ayarlı

enum **PriceFields**: ***Engine.GetPriceList*** fonksiyonunda kullanılır.
enum **PriceTypes**: ***SendOrder*** fonksiyonunda kullanılan parametredir. *Market* ve *Limit* olmak üzere iki tanımı vardır. *Market* seçildiğinde piyasa, *Limit* seçildiğinde limit emir iletir.
enum **OrderStatus**: Emrin durumunu belirtir.

Waiting: Bekleyen
Delivered: Teslim Edildi
Filled: Gerçekleşti
Partial: Kısmi Gerçekleşti
Canceled: İptal Edildi
Replaced: Değiştirildi
Suspended: Askıya Alındı
Expired: Süresi Doldu
Error: Hata

SİSTEM FONKSİYONLARI

- **Fiyat Verisine Abone Olma Fonksiyonu**
  public void ***SubscribePrice****(string symbol)*

Stratejinizde **Load** fonksiyonunda kullanılır. Seçilen sembolün fiyat verilerine abone olmak için kullanılır. Bu fonksiyonu kodunuzda bir defa çağırmanız yeterlidir.

symbol: Büyük harflerle sembolün adı string olarak verilmelidir.

- **Derinlik Verisine Abone Olma Fonksiyonu**
  public void ***SubscribeDepth****(string symbol)*

Stratejinizde **Load** fonksiyonunda kullanılır. Seçilen sembolün derinlik verilerine abone olmak için kullanılır. Bu fonksiyonu kodunuzda bir defa çağırmanız yeterlidir.

symbol: Büyük harflerle sembolün adı string olarak verilmelidir.

- **Bar Getirme Fonksiyonu**
  *public List<****Bar****>* ***GetCandles****(string symbol, string period)*

Stratejinizde sembolün verisini kullanmak istiyorsanız bu fonksiyon ile o sembole ait verileri çekebilirsiniz. Bu fonksiyonu kodunuzun başında bir defa çağırmanız yeterlidir.

symbol: Büyük harflerle sembolün adı string olarak verilmelidir.
period: Dakika cinsinden periyot string olarak verilmelidir.

Geri dönüş sağlayan **Bar** modeli aşağıdaki gibidir.

```
{
    double Open,        :Açılış
    double Close,       :Kapanış
    double High,        :Yüksek
    double Low,         :Düşük
    double Amount,      :Miktar
    double Volume,      :Hacim
    DateTime Date       :Tarih
}
```

- **Derinlik Getirme Fonksiyonu**
  *public Dictionary<int,****Depth****>* ***GetDepths****(string symbol)*

Stratejinizde sembolün derinlik verisini kullanmak istiyorsanız bu fonksiyon ile o sembole ait derinlik verilerini çekebilirsiniz. Bu fonksiyonu kodunuzun başında bir defa çağırmanız yeterlidir. Parametre olarak büyük harflerle sembolün adını vermeniz yeterlidir.

Geri dönüş sağlayan **Depth** modeli aşağıdaki gibidir.

```
{
    string Symbol,          :Sembol
```

string **Market,** :Pazar Tipi
string **Direction,** :Yön
int **Row,** :Satır
int **Quantity,** :Lot Miktarı
double **Price,** :Fiyat
int **OrderCount,** :Emir Miktarı
DateTime **Date** :Tarih
}

- **Sembole Ait Basit Verileri Getirme Fonksiyonu**
  *public **BasicData** **GetBasicData**(string symbol)*

Stratejinizde sembolün basit verilerini kullanmak istiyorsanız bu fonksiyon ile o sembole ait basit verilerini çekebilirsiniz. Bu fonksiyonu kodunuzun başında bir defa çağırmanız yeterlidir. Parametre olarak büyük harflerle sembolün adını vermeniz yeterlidir.

Geri dönüş sağlayan **BasicData** modeli aşağıdaki gibidir.
{
string **Symbol,** :Sembol
string **Definition,** :Şirket Adı
double **LastPrice,** :Son Fiyat
double **Difference,** :Değişim
double **DifferencePercent,** :Değişim Yüzde
double **PreviousDay,** :Önceki Gün Açılış Fiyatı
double **PreviousWeek,** :Bu Hafta Kapanış Fiyatı
double **PreviousOneWeek,** :1 Hafta Kapanış Fiyatı
double **PreviousMonth,** :Bu Ay Kapanış Fiyatı
double **PreviousOneMonth,** :1 Ay Kapanış Fiyatı
double **PreviousYear,** :Bu Yıl Kapanış Fiyatı
double **PreviousOneYear,** :1 Yıl Kapanış Fiyatı
double **FK,** :Fiyat/Kazanç
double **FreeFloatRate,** :Açıklık
double **MarketBookValueRatio,** :
long **MarketValue,** :Piyasa Değeri
double **BookValue,** :Defter Değeri
string **BalanceSheetDate,** :Dönem
long **Capital,** :Ödenmiş Sermaye
long **EquityCapital,** :Öz Sermaye
long **NetProfit,** :Net Kar
long **ShareCount,** :Senet Sayısı
long **Volume,** :Hacim
double **Open,** :Açılış
double **High,** :Yüksek
double **Low,** :Düşük
double **Average**, :A.Ort

```
        DateTime Date,                  :Tarih
        string Group,                   : Pazar Grubu
        string Sector                   : Sektör
    }
```

- **Emir Gönderme**

  *public **string** **SendOrder**(string symbol, Directions direction, double lot, PriceTypes pricetype, double price = 0)*

Emir göndermek için bu fonksiyon kullanılır. İlk üç parametresi zorunlu olmakla birlikte emir tipi, emir fiyatı gibi değişkenler isteğe bağlı olarak ayarlanabilir.

symbol: Emrin gönderileceği sembol string olarak verilmelidir.
direction: Emrin yönü birkaç farklı şekilde verilebilir; **Directions.BUY** veya **Directions.SELL**
lot: Emrin kaç lot olacağı double olarak verilmelidir.
pricetype: **PriceTypes.Market** veya **PriceTypes.Limit** olmalıdır. Limit emir verilmesi durumunda sonraki parametrede fiyat bilgisi girilmelidir.
price: Limit emirlerde emrin double cinsinden fiyatını belirler.

Geri dönüş olarak gönderilen emrin **ID**'sini döner.

- **Emir İyileştirme**

  *public void **ModifyOrder**(string id, double lot, double price)*

Daha önceden gönderilen emri iyileştirmek için kullanılan fonksiyondur.

id: Emri gönderirken geri dönen ID verilmelidir.
lot: Emrin kaç lot olacağı double olarak verilmelidir.
price: Limit emirlerde emrin double cinsinden fiyatını belirler.

- **Emir İptal Etme**

  *public void **CancelOrder**(string id)*

Daha önceden gönderilen emri iptal etmek için kullanılan fonksiyondur.

id: Emri gönderirken geri dönen ID verilmelidir.

- **Pozisyonlari Kapat**

  *public **string** **ClosePositions**(string Symbol)*

Bu metot stratejinizin o anki pozisyon bilgisini okuyarak ters yönde aynı miktarda emir gönderir. Böylece stratejiye ait açık pozisyonlarınız sıfırlanmış olur.

Symbol: Emrin gönderileceği sembol string olarak verilmelidir.

- **Log Yazdırma**

*public void **SendMessage**(MessageTypes type, string message)*

Bu metot stratejiniz çalışırken istediğiniz metni log olarak yazdırma için kullanılan fonksiyondur.

type: Üç çeşit mesaj var.

- MessageTypes.Command : Bu parametre seçildiğinde message parametresine "**STOP**" yazılmalıdır. Bu şekilde çalıştığında stratejinizi durdurma işlemi yapmış olursunuz.
- MessageTypes.Log : Bu parametre seçildiğinde message parametresine istediğiniz string değerini yazarak log yazdırma işlemi kullanılır.
- MessageTypes.Notification : Bu parametre seçildiğinde message parametresine istediğiniz string değeri yazarak mobil platforma bildirim gönderilmesi sağlanır.

message: İstenilen mesajı string olarak yazabilirsiniz.

- **Brüt Takas Kontrol Fonksiyonu**

  *public **bool** **GetBlockedSymbol**(string symbol)*

Bu metot stratejide çalıştırmış olduğunuz sembol' ün brüt takasta olup olmadığını kontrol eder. Eğer girilen sembol brüt takasta ise **true** değeri döndürür.

symbol: Emrin gönderileceği sembol string olarak verilmelidir.

ENGİNE KÜTÜPHANESİ FONKSİYONLARI

- **GetPriceList**
  *public **List<double>** GetPriceList(List<Bar> candles, PriceFields type)*

  Bu metot stratejide çalıştırmış olduğunuz sembol' e ait çekmiş olduğunuz barlardan seçilen fiyat listesini getiren fonksiyondur.

  candles: **GetCandles** metodundan dönen Bar listesini alır.
  type: PriceFields tipinde değişken alır.

- **LastPrice**
  *public **double** LastPrice(List<Bar> candles)*

  Bu metot stratejide çalıştırmış olduğunuz sembol' e ait çekmiş olduğunuz barlardan son fiyatı getiren fonksiyondur.

- **LastValue**
  *public **double** LastValue(List<double> values)*

  Bu metot stratejide çalıştırdığınız herhangi bir double listedeki son değeri döndüren fonksiyondur.

- **PreviousValue**
  *public **double** PreviousValue(List<double> values, int back)*

  Bu metot stratejide çalıştırdığınız herhangi bir double listedeki back parametresine yazılan önceki değerini döndüren fonksiyondur.

- **GetLine**
  *public **List<double>** GetLine(List<List<double>> values, int index)*

  Bu metot stratejinizde bir double iç içe listteki seçilen index' e göre double cinsinden list getiren fonksiyondur.

- **Average**
  *public **double** Average(List<double> values)*

  Bu metot girilen listenin ortalamasını alan fonksiyondur.

- **StandardDeviation**
  public **double** StandardDeviation(List<double> values)

  Bu metot girilen listenin standart sapmasını alan fonksiyondur.

- **HighestValue**
  public **double** HighestValue(List<double> values)

Bu metot girilen liste içerisindeki en yüksek değeri döndüren fonksiyondur.

- **LowestValue**
  public **double** LowestValue(List<double> values)

  Bu metot girilen liste içerisindeki en düşük değeri döndüren fonksiyondur.

- **HighestHigh**
  public **double** HighestHigh(List<double> datalist, double periodX)

  Bu metot girilen **datalist** listesinde sondan **periodX** kadarı içerisindeki en yüksek değeri döndüren fonksiyondur.

- **LowestLow**
  public **double** LowestLow(List<double> datalist, double periodX)

  Bu metot girilen **datalist** listesinde sondan **periodX** kadarı içerisindeki en düşük değeri döndüren fonksiyondur.

- **Intersect**
  public **bool** Intersect(List<double> values1, dynamic values2, string direction)

  Bu metot girilen **values1** listesini **values2** değeri **direction** yönündeki kesimini kontrol eden fonksiyondur. Eğer kesişim var ise **true** cevabını dönmektedir.

  values1: Kesişimin kontrol edileceği liste
  values2: Int, Double, List gibi değişken bir parametredir.
  direction: Yön olarak tanımlanır. Örn: yukarı,aşağı,up,down

- **AbsoluteValue**
  public double AbsoluteValue(double value)

  Bu metot girilen değerin mutlak değerini döndüren fonksiyondur.

- **Cube**
  public **double** Cube(double value)

  Bu metot girilen değerin küpünü döndüren fonksiyondur.

- **ArcTanjant**
  public **double** ArcTanjant(double value)

  Bu metot girilen değerin ark tanjantını döndüren fonksiyondur.

- **Exponent**
  public **double** Exponent(double value, double multiplier)

Bu metot girilen **value** değerinin **multiplier** değerinde üssünü döndüren fonksiyondur.

- **SquareRoot**
  public **double** SquareRoot(double value)

  Bu metot girilen değerin karekökünü döndüren fonksiyondur.

- **Sinus**
  public **double** Sinus(double value)

  Bu metot girilen değerin sinüsünü döndüren fonksiyondur.

- **Cosinus**
  public **double** Cosinus(double value)

  Bu metot girilen değerin kosinüsünü döndüren fonksiyondur.

- **Tanjant**
  public **double** Tanjant(double value)

  Bu metot girilen değerin tanjantını döndüren fonksiyondur.

ÖRNEK KODLAMALAR

- **İlk Stratejim**

Bir sembolde belirli bir fiyat seviyesinin altına indiğinde alış emri, belirli bir fiyat seviyesinin üzerine çıktığında satış emri gönderecek bir strateji yazalım.

```
//Genel parametreleri burada tanımlıyoruz.
public string Sembol="GARAN";
public string SonYon = "";
public int lot = 1;
public double alısseviye = 2.06;
public double satısseviye = 2.16;

public void Load()
{
    //Fiyata abone olma Fonksiyonu
    SubscribePrice(Sembol);
}

public void PriceChanged(Tick t)
{
    if(t != null && t.Price!=0)
    {
        if(t.Price <= alısseviye && SonYon == "")
        {
            //ALIŞ koşulu gerçekleşti, SonYon değerimizi eşitliyoruz.
            SonYon = "A";
            //Emir gönder fonksiyonu ile emri belirlediğimiz “lot” miktarında gönderiyoruz.
            SendOrder(Sembol,Directions.BUY,lot,PriceTypes.Market);
            //Log Yazdırma
            SendMessage(MessageTypes.Log,"Alış: "+t.Price.ToString());
        }
        else if(t.Price >= satısseviye && SonYon == "A")
        {
            //SATIŞ koşulu gerçekleşti, SonYon değerimizi eşitliyoruz.
            SonYon = "";
            //Emir gönder fonksiyonu ile emri belirlediğimiz “lot” miktarında gönderiyoruz.
            SendOrder(Sembol,Directions.BUY,lot,PriceTypes.Market);
            //Log Yazdırma
            SendMessage(MessageTypes.Log,"Satış: "+t.Price.ToString());
        }
    }
}
```

- **İndikatör Kullanmak**

  Stratejimizde indikatör kullandığımız bir algoritama yazalım. Hareketli ortalamanın kesişimine göre çalışan bir strateji yazalım. Buna göre Hareketli ortalama değeri 30 değerini aşağıdan yukarı keserse “ALIŞ”, 70 değerini yukarıdan aşağı keserse “SATIŞ” yapalım.

```
//Genel parametreleri burada tanımlıyoruz.
public string Sembol="GARAN";
public string Periyot="5";
public string SonYon = "";
public int rsiperiod =1;
public int lot=1;

public void Load()
{
  //Fiyata abone olma Fonksiyonu
  SubscribePrice(Sembol);
}

public void PriceChanged(Tick t)
{
  //Bar Getirme Fonkstiyonu
  var candles = GetCandles(Sembol,Periyot);
  //Barlardaki Kapanış Fiyatlarını Getirme
  var C = Engine.GetPriceList(candles, PriceFields.Close);
  //RSI indikatörünü oluşturan fonksiyon
  var RSI = Engine.RSI(C, rsiperiod);

  //Alış Koşulu
  if( Engine.PreviousValue(RSI, 1) < 30 && Engine.PreviousValue(RSI, 2) > 30 && SonYon == "")
  {
    SonYon = "A";
    SendOrder(Sembol,Directions.BUY,lot,PriceTypes.Market);
  }
  //Satış Koşulu
  else if(Engine.PreviousValue(RSI,1) > 70 && Engine.PreviousValue(RSI, 2) < 70 && SonYon
== "A")
  {
    SonYon = "";
    SendOrder(Sembol,Directions.SELL,lot,PriceTypes.Market);
  }
}

public void OrderStatusChanged(Order o)
{
}
public void DepthChanged(Depth d)
{
}
```

- **İndikatör Kullanmak 2**
  Bazı indikatörlerde birden fazla çizgi bulunur. Örneğin Bollinger Bantları Üst, Orta ve Alt bantlar olmak üzere 3 çizgiye sahiptir. Bollinger Bantları kullanan bir strateji oluşturalım.

```
//Genel parametreleri burada tanımlıyoruz.
public string Sembol = "GARAN";
public string Periyot = "1";
public string SonYon = "";
public int rsiperiod = 1;
public int lot = 1;

public void Load()
{
  //Fiyata Abone Olma Fonksiyonu
  SubscribePrice(Sembol);
}

public void PriceChanged(Tick t)
{
  //Bar Getirme Fonksiyonu
  var candles = GetCandles(Sembol,Periyot);
  //Barlardaki Kapanış Fiyatlarını Getirme Fonksiyonu
  var C = Engine.GetPriceList(candles, PriceFields.Close);
  //Bollinger İndikatörünü Oluşturma Fonksiyonu
  var bollinger = Engine.BollingerBands(candles,14,2);

  //Bollinger çizgilerinden hangisini almak istediğimizi aşağıdaki gibi belirteceğiz.
  //Alt çizgi
  List<double> altBollinger = bollinger[2];
  //Orta çizgi
  List<double> ortaBollinger = bollinger[1];
  //Üst çizgi 0 verilmesinin sebebi ilk sıranın her zaman 0'dan başlamasıdır.
  List<double> ustBollinger = bollinger[0];

  //Üst çizginin kapanmış son mumdaki değerini alalım.
  double ustSonDeger = Engine.PreviousValue(ustBollinger, 1);
  //Alt çizginin kapanmış son mumdaki değerini alalım.
  double altSonDeger = Engine.PreviousValue(altBollinger, 1);
  //Bir önceki kapanış değerimizi alalım.
  double sonKapanis = Engine.PreviousValue(C, 1);
  //Kapanış değeri üst çizginin üstünde kalırsa ALIŞ yapalım
  if(sonKapanis > ustSonDeger && SonYon == "")
  {
    SonYon = "A";
    SendOrder(Sembol,Directions.BUY,lot,PriceTypes.Market);
  }
  //Kapanış değeri alt çizginin altında kalırsa SATIŞ yapalım
  else if(sonKapanis < altSonDeger && SonYon == "A")
```

```
    {
        SonYon = "S";
        SendOrder(Sembol,Directions.SELL,lot,PriceTypes.Market);
    }
}
public void OrderStatusChanged(Order o)
{
}
public void DepthChanged(Depth d)
{
}
```

- **Zamana Bağlı Strateji**

  Stratejilerde zamana bağlı koşullar yazılabilir. Örneğin saat 17 sonrasında işlemleri kapatacak bir koşul yazabiliriz. Zamana bağlı strateji kullanırken iki farklı tarihten yola çıkabiliriz. Birincisi sistem saatini kullanarak o anki saati elde etmek. İkincisi ise mum grafiğindeki son mum çubuğunun tarihini alarak en yakın saati elde etmek.

```
//Genel parametreleri burada tanımlıyoruz.
public string Sembol = "TSKB";
public string Periyot = "5";
public string SonYon = "";
public int SonPozisyon = 0;
public int lot = 500;

public void Load()
{
  //Fiyata Abone Olma Fonksiyonu
  SubscribePrice(Sembol);
}
public void PriceChanged(Tick t)
{
  var candles = GetCandles(Sembol,Periyot);
  var C = Engine.GetPriceList(candles, PriceFields.Close);
  //RSI indikatörünün 30 ve 70 seviyelerini geçmesi durumunda işlem yapan ve saat 17'de pozisyonları kapatan bir strateji yazalım.
  var rsi = Engine.RSI(C,14);
  var rsideger1 = Engine.PreviousValue(rsi,1);
  //Stratejinin çalıştığı mum grafiğindeki son mum tarihini alalım.
  DateTime sonMumTarihi = candles[candles.Count - 1].Date;
  //Son mum barının saati 17'den küçükse ALIŞ veya SATIŞ yapılabilsin.
  if(sonMumTarihi.Hour < 17)
  {
    if(rsideger1 > 70 && SonYon == "")
    {
      SonYon = "A";
      SendOrder(Sembol,Directions.BUY,Engine.AbsoluteValue(SonPozisyon)+ lot, PriceTypes.Market);
      SonPozisyon=lot*1;
    }
    if(rsideger1 < 30 && SonYon == "A")
    {
      SonYon = "";
      SendOrder(Sembol,Directions.SELL,Engine.AbsoluteValue(SonPozisyon)+ lot,PriceTypes.Market);
      SonPozisyon=lot*(-1);
    }
  }
  //Son mum barının saati 17 ise işlemler kapansın.
  if(sonMumTarihi.Hour == 17 && SonYon != "F")
```

```
    {
        //Pozisyonları kapat ve son yönü flat yap.
        SonYon = "F";
        ClosePositions(Sembol);
    }
}
public void OrderStatusChanged(Order o)
{
}
public void DepthChanged(Depth d)
{
}
```

- **Çoklu Sembol Kullanılan Strateji**
  Birden fazla sembol kullanarak fiyata göre işlem yapan strateji yazalım

```
public List<string> Symbols=new List<string>(){"AGHOL","AKBNK"};
public List<double> AlısFiyatList = new List<double>();
public List<double> AlınanLotList = new List<double>();
public List<string> SonYonList =new List<string>();
public List<double> Fiyatlar = new List<double>();

public string Period="1";
public double Amount=50;

public void Load()
{
  //Listelerin eleman sayısını sembolün eleman sayısına eşitleme işlemleri
  AlısFiyatList=new List<double>(new double[Symbols.Count]);
  AlınanLotList=new List<double>(new double[Symbols.Count]);
  SonYonList=new List<string>(new string[Symbols.Count]);
  Fiyatlar=new List<double>(new double[Symbols.Count]);

  foreach(var item in Symbols)
  {
    SubscribePrice(item);
  }
}
public void PriceChanged(Tick t)
{
  if(t!=null)
  {
    var index=Symbols.IndexOf(t.Symbol);

    if(index >= 0)
    {
      if(Fiyatlar[index]!=t.Price)
      {
        Fiyatlar[index]=t.Price;
        //GetBlockedSymbol fonksiyonu sembol brüttakasta ise true değilse false döndürür.
        if(!GetBlockedSymbol(Symbols[index]) && t.Price != 0 )
        {
          //Barlar listesini getirme
          var candles=GetCandles(Symbols[index], Period, 10);
          // Kapanış Fiyatlarının Listesini getirme
          var C=Engine.GetPriceList(candles, PriceFields.Close);

          // Alış Koşulu Örnek
          if(Fiyatlar[index]>t.Price && SonYonList[index]=="")
          {
            SonYonList[index]="A";
```

```
                AlısFiyatList[index]=t.Price;
                //Log yazdırma
                SendMessage(MessageTypes.Log, "İlk Alış "+t.Symbol+": "+ t.Price.ToString());

                int lot=1;
                if(Amount/t.Price>1)
                {
                   lot=Convert.ToInt32(Math.Floor(Amount/t.Price));
                }
                AlınanLotList[index]=lot;
                //Emir Gönderim fonksiyonu
                SendOrder(Symbols[index], Directions.BUY, lot, PriceTypes.Limit,t.Price);
                //Log yazdırma
                SendMessage(MessageTypes.Log, "İlk Alış "+t.Symbol+" : " +
AlısFiyatList[index].ToString());
            }


            // Satış Koşulu Örnek
            else if(t.Price > AlısFiyatList[index] && SonYonList[index]=="A")
            {
                SonYonList[index]="";
                //log Yazdırma
                SendMessage(MessageTypes.Log, "Kar Koşulu "+t.Symbol+": "+t.Price.ToString()
+ " : " + AlısFiyatList[index].ToString()+":"+SonYonList[index].ToString());
                //Emir Gönderim
                SendOrder(Symbols[index], Directions.SELL, AlınanLotList[index],
PriceTypes.Limit,t.Price);
            }
          }
        }
      }
   }
}
public void OrderStatusChanged(Order o)
{

}
```

- **Emiri Dinleyerek İşlem Yapma**

  Emir gönderildikten sonra emrin durumunu dinleyip gerçekleşti ise diğer işlemi yapan bir strateji yazalım.

```
/*Parametreler*/
public string Sembol="GARAN";
public string Periyot="5";
public int lot=1;
public double alınanfiyat = 0;
public string SonYon = "";
/*Parametreler*/
public void Load()
{
  //Fiyata Abone Olma Fonksiyonu
  SubscribePrice(Sembol);
  //Emir Gerçekleştiğinde Email Gönderimi
  EmailNotification=true;
  //Emir Gerçekleştiğinde Sms Gönderimi
  SmsNotification=true;
}
public void PriceChanged(Tick t)
{
  double Kademe=0;
  if(t.Price<20)
  {
    Kademe=0.01;
  }
  else if(t.Price>=20 && t.Price<50)
  {
    Kademe=0.02;
  }
  else if(t.Price>=50 && t.Price<100)
  {
    Kademe=0.05;
  }
  else if(t.Price>=100)
  {
    Kademe=0.1;
  }
  //Bar Getirme Fonksiyonu
  var candles = GetCandles(Sembol,Periyot);
  //Bar içerisindeki Kapanış Fiyatlarını Getirme Fonksiyonu
  var C = Engine.GetPriceList(candles, PriceFields.Close);
  //Alış Koşulu
  if(Engine.PreviousValue(C,2) > Engine.PreviousValue(C, 1) && Engine.PreviousValue(C, 1) >
t.Price  && SonYon == "")
  {
    //Log Yazdırma
    SendMessage(MessageTypes.Log, "ALIŞ girildi: " + t.Price.ToString());
```

```
        SonYon = "K";
        //Emir Gönderim
        SendOrder(Sembol, Directions.BUY, lot, PriceTypes.Limit, t.Price-Kademe);
    }
    //Satış Koşulu
    else if(t.Price > alınanfiyat && alınanfiyat != 0 && SonYon == "A"){
        //Log Yazdırma
        SendMessage(MessageTypes.Log, "SATIŞ girildi: " + t.Price.ToString());
        SonYon = "Z";
        //Emir Gönderim
        SendOrder(Sembol, Directions.SELL, lot, PriceTypes.Limit, t.Price+Kademe);
    }
}
public void OrderStatusChanged(Order o){
    //Alış Emri Gerçekleşme Kontrolü
    if(o.Status == OrderStatus.Filled && o.Direction == Directions.BUY && SonYon=="K"){
        SendMessage(MessageTypes.Log, "ALIŞ gerçekleşti.");
        alınanfiyat=o.Price;
        SonYon = "A";
    }
    //Alış Emri Hatalı İse
    else if(o.Status == OrderStatus.Error && o.Direction == Directions.BUY && SonYon=="K"){
        SendMessage(MessageTypes.Log, "ALIŞ gerçekleşmedi. "+o.Comment.ToString());
        SonYon = "";
    }
    //Alış İptal Olma Kontrolü
    else if((o.Status == OrderStatus.Expired || o.Status == OrderStatus.Canceled) &&
o.Direction == Directions.BUY && SonYon=="K"){
        SendMessage(MessageTypes.Log, "ALIŞ gerçekleşmedi.");
        SonYon = "";
    }
    //Satış Emir Gerçekleşme Kontrolü
    else if(o.Status == OrderStatus.Filled && o.Direction == Directions.SELL && SonYon=="Z"){
        SendMessage(MessageTypes.Log, "SATIŞ gerçekleşti.");
        alınanfiyat=0;
        SonYon = "";
    }
    //Satış Emri Hatalı İse
    else if(o.Status == OrderStatus.Error && o.Direction == Directions.SELL && SonYon=="Z"){
        SendMessage(MessageTypes.Log, "SATIŞ gerçekleşmedi. "+o.Comment.ToString());
        SonYon = "A";
    }
    //Satış İptal Olma Kontrolü
    else if((o.Status == OrderStatus.Expired || o.Status == OrderStatus.Canceled) &&
o.Direction == Directions.SELL && SonYon=="Z"){
        SendMessage(MessageTypes.Log, "SATIŞ gerçekleşmedi.");
        SonYon = "A";
    }}
```