KARADENİZ TEKNİK ÜNİVERSİTESİ
İKTİSADİ VE İDARİ BİLİMLER FAKÜLTESİ
YÖNETİM BİLİŞİM SİSTEMLERİ BÖLÜMÜ

YBS4009 NESNE TABANLI PROGRAMLAMA DÖNEM PROJESİ

Hazırlayanlar

Metehan TUNCER
Mustafa Mert KISA
Utku GEDİK

Danışman

Dr. Öğr. Üyesi Mustafa Bilgehan İMAMOĞLU

Aralık 2021

ÖZET

Dilimize Nesne yönelimli programlama olarak çevirdiğimiz ve orijinali Object Oriented Programming ifadesi özetle bir bilgisayar programlama yaklaşımıdır. 1960'lı yılların sonunda ortaya çıkan bu yaklaşım o dönem yazılım geliştirmede ortaya çıkan sorunları ortadan kaldırmak için ortaya atılmıştır. Nesne yönelimli programlama faktörleriyle hazırlanmış olan bu proje de, banka süreçleri çalışır hâle getirilmiştir. C# yazılım diliyle hazırlanan uygulama:

- Hesap açma
- Para yatırma
- Para çekme
- Hesaplar listesini görüntüleme
- Hesap durumlarını inceleme
- Hesap işlem kayıtlarını gözlemleme
- Mevcut hesaplar arasından çekiliş yaparak bir hesabı seçebilme

faaliyetlerini yerine getirebilmektedir. Uygulama her başlatıldığında kullanıcıdan TC Kimlik Numarası bilgisi alınır ve ilgili fonksiyona iletilerek cari hesabı oluşturulmuş olur. Daha sonrasında kullanıcı yatırma ve çekim işlemlerini bu vadesiz olan cari hesabı üzerinden yürütür. İsteği doğrultusunda ana menüde yer alan hesap açma seçeneği ile vadeli hesap seçeneklerini ve hesapların gerektirdiği şartları inceleyebilir. Vadeli hesap açmak istediğinde tercihini yapar ve hesabının oluşturulduğunu konsol üzerinde gözlemleyebilir. Aynı zamanda hesaplar listesi veya hesap durumunda da ilgili hesaplarının detaylarına ulaşabilecektir. Hesap işlem kayıtları aracılığıyla da uygulama içerisinde tamamladığı işlemlerin listesini gözlemleyebilmektedir.

TANIMLAR

1. Program.cs

➔ Main Fonksiyonu

Main fonksiyonu içerisinde kullanıcıdan TC Kimlik Numarası bilgisi alınır. Daha sonrasında *BankAccount* sınıfından yeni bir nesne oluşturur ve TC Kimlik Numarasını nesneye ait olan *LoginAccount* fonksiyonuna iletir.

```
            var identityNumber = Console.ReadLine();
            var acc = new BankAccount();
            acc.LoginAccount(acc, identityNumber);
```

Kullanıcıdan menü üzerinde işlem tercihi yapması istenir. *menuKey* değişkenine kullanıcının girdi olarak verdiği menü tercihi tanımlanır. While döngüsünde bu tercihin çıkış yap olmadığı durumlar için koşullar çalıştırılır.

```
            var menuKey = Console.ReadLine();
            while (menuKey != "8")
            {
                if (menuKey == "1")
                {
                    Console.WriteLine("\n1 = Kısa Vadeli Hesap [Yıllık kâr oranı: %15 | Min gereken
tutar: 5000 TL]\n**************************************\n" + "2 = Uzun Vadeli Hesap [Yıllık kâr oranı:
%17 | Min gereken tutar: 10000 TL]\n**************************************\n" + "3 = Özel Hesap
[Yıllık kâr oranı: %10 | Min gereken tutar: 0 TL]");
                    var amount = Console.ReadLine(); //Burada amount'u hesap türü tercihi olarak
almaktayız

                    MakeOperation(acc, int.Parse(menuKey), amount);
                    ShowDialogMenu(0); //Ana menü
                }
                ...
            }
```

Benzer şekilde devam eden if komutlarıyla *menuKey*'in değerine uygun olacak işlevi yerine getirmesi gereken fonksiyonlar tetiklenmektedir.

```
                else if (menuKey == "4")
                {
                    var amount = "AccList";
                    MakeOperation(acc, int.Parse(menuKey), amount);
                    ShowDialogMenu(0);
                }
```

➔ ShowDialogMenu Fonksiyonu

ShowDialogMenu gelen *menuKey* değişkenine uygun olacak şekilde ilk konsola çıktıyı veren fonksiyondur. Bu fonksiyonun uygulamada ki en önemli göre case 0'da yer alan ana menüdür. *Main* fonksiyonu içerisinde yer alan hareketlerden sonra *ShowDialogMenu(0)* komutu verilir. Böylelikle gerçekleşen işlemlerden sonra kullanıcı ana menüye yönlendirilmiş olur.

```
public static void ShowDialogMenu(int menuKey)
        {
            switch (menuKey)
            {
                case 0:
                    Console.WriteLine("\n1 = Hesap Açma\n********************\n"+"2 = Para
Yatırma\n********************\n"+"3 = Para Çekme\n********************\n"+"4 = Hesap
Listesi\n********************\n"+"5 = Hesap Durum\n******************** \n"+"6 = Hesap İşlem
Kayıtları\n********************\n"+"7 = Çekiliş\n********************\n"+"8 = Çıkış Yap");
                    break;
                case 1:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Hesap türü seçiniz:");
                    Console.WriteLine("#######################################");
                    break;
                case 2:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Yatırmak istediğiniz tutarı giriniz:");
                    Console.WriteLine("#######################################");
                    break;
                case 3:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Çekmek istediğiniz tutarı giriniz:");
                    Console.WriteLine("#######################################");
                    break;
                case 4:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Hesap listesi:");
                    Console.WriteLine("#######################################");
                    break;
                case 5:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Hesap durum:");
                    Console.WriteLine("#######################################");
                    break;
                case 6:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Hesap işlem kayıtları:");
                    Console.WriteLine("#######################################");
                    break;
                case 7:
                    Console.WriteLine("#######################################");
                    Console.WriteLine("Çekiliş:");
                    Console.WriteLine("#######################################");
                    break;
                default:
                    break;
            }
        }
```

➔ MakeOperation Fonksiyonu

Tetiklemeler için *MakeOperation* fonksiyonu kullanılmaktadır. BankAccount sınıfından oluşturduğumuz *acc* nesnesi ile birlikte *menuKey*'in ve *amount*'un parametre olarak iletilmesi gerekmektedir. Diğer tercih durumlarının kullanıldığı kısımlar gibi burada da switch case yapısı kullanılmıştır. Aşağıda yer alan kod bloğunda hesap açma tercihinin yapıldığı ve kısa vadeli hesap istendiği durumun operasyonu incelenebilir.

```
 switch (menuKey)
            {
                case 1:
                    if (amount == "1" && acc.CheckingAccounts.Balance >= 5000)
                    {
                        Console.WriteLine("Oluşturacağınız hesabınıza aktarmak istediğiniz tutarı
giriniz:");
                        var startAmount = Console.ReadLine();
                        acc.CreateAccount(acc, amount, startAmount);
                        break;
                    }
```

Cari hesapta yer alan bakiyenin, tercih edilen hesabın açılabilmesi için yeterli olup olmadığı kontrol edildikten sonra eğer koşul sağlanıyorsa, vadeli hesap için kullanıcının aktarmak istediği tutar alınır ve *startAmount* değişkenine tanımlanır. Daha sonrasında *acc* nesnesinin *CreateAccount* fonksiyonuna *acc*, *amount* ve *startAmount* parametreleri ile birlikte iletilir.

Kullanıcı tercihine göre para yatırma, para çekme ve diğer işlevler için aynı yöntem izlenir sadece acc nesnesi içerisinde izlenen fonksiyonlar değişmektedir.

```
                case 2:
                    acc.DepositMoney(acc, amount);
                    Console.WriteLine("########################################");
                    Console.WriteLine("Bakiye eklendi.");
                    Console.WriteLine("########################################");
                    break;
                case 3:
                    if (acc.CheckingAccounts.Balance < int.Parse(amount))
                    {
                        Console.WriteLine("#####################################");
                        Console.WriteLine("Yetersiz bakiye.");
                        Console.WriteLine("#####################################");
                        break;
                    }
                    acc.WithdrawMoney(acc, amount);
                    break;
```

2. BankAccount.cs

➔ BankAccount Sınıfı

Sınıf içerisinde uygulamamızda yer alan hesap türlerinin sınıfları oluşturulur. Hesap adı Enum olarak tanımlanmaktadır. Sınıfın diğer parçaları bakiye, minimum gereken bakiye, kâr oranı ve hesap numarasıdır.

```
public class ShortDepositAccount //Kısa vadeli hesap
{
    public AccEnum AccountName { get; set; }
    public decimal Balance { get; set; }
    public int MinRequiredMoney { get; set; }
    public int IncomeRatio { get; set; }
    public int AccountNumber { get; set; }
}

public class SpecialDepositAccount //Özel hesap
{
    public AccEnum AccountName { get; set; }
    public decimal Balance { get; set; }
    public int MinRequiredMoney { get; set; }
    public int IncomeRatio { get; set; }
    public int AccountNumber { get; set; }
}

public class LongDepositAccount //Uzun vadeli hesap
{
    public AccEnum AccountName { get; set; }
    public decimal Balance { get; set; }
    public int MinRequiredMoney { get; set; }
    public int IncomeRatio { get; set; }
    public int AccountNumber { get; set; }
}

public class CheckingAccount //Cari hesap
{
    public AccEnum AccountName { get; set; }
    public decimal Balance { get; set; }
    public int AccountNumber { get; set; }
}
```

Uygulamanın ilerleyen aşamalarında gerçekleştirilecek adımlar için bu sınıflardan oluşturulacak nesneler ile işlemler yapılacaktır. İşlemler için *BankAccount* sınıfı altında birçok fonksiyon tanımlanmıştır.

➔ LoginAccount Fonksiyonu

Kullanıcı ilk uygulamayı başlattığında çalışacak olan fonksiyondur. TC Kimlik Numarasını ve *account* nesnesini parametre olarak alır. Daha sonrasında *Random* fonksiyonu ile 8 haneli hesap numarasını oluşturur. *CheckingAccount* sınıfına ait olan nesneye hesap adı, bakiye, hesap numarası gibi tanımlamalar yapılır. Henüz kullanıcıdan bir bakiye bilgisi almadığımız için 0 olarak tanımlamaktayız.

```
  public void LoginAccount(BankAccount account, string identityNumber)
        {
            //Hesap no oluşturma
            Random rnd = new Random();
            int randomNo = rnd.Next(10000000, 99999999);

            account.CheckingAccounts = new CheckingAccount();
            account.IdentityNumber = identityNumber;
            account.CheckingAccounts.Balance = 0;
            account.CheckingAccounts.AccountName = AccEnum.Cari;
            account.CheckingAccounts.AccountNumber = randomNo;

            accountList.Add($"\nHesap adı & numarası: {account.CheckingAccounts.AccountName} -
{account.CheckingAccounts.AccountNumber}"); //Hesap listesi

            checkingInfoName = account.CheckingAccounts.AccountName;
            checkingInfoNumber = account.CheckingAccounts.AccountNumber;
            checkingInfoBalance = account.CheckingAccounts.Balance;
        }
```

Hesapları listeleyeceğimiz özellik için hesapları listede tutuyoruz. Bunun için her hesap oluşturulduğunda hesaba ilişkin hesap adı ve hesap numarası bilgileri *accountList* isimli listeye eklenmektedir.

```
List<string> accountList = new List<string>(); //Hesap listesi
```

Hesap durumunda gösterilecek hesap bilgileri için nesnelerden çağrılan verilerin kullanılması eğer o nesne henüz tanımlanmamışsa, yani o hesap türünden oluşturulmamışsa sorun yaratabilmektedir. Bu nedenle bu sorunun yaşanmaması adına hesap durum için kullanacağımız verileri *checkingInfoName, checkingInfoNumber* ve *checkingInfoBalance* değişkenlerine tanımlıyoruz. Hesap isimleri, hesap türünü belirtmektedir ve hesap türleri *AccEnum'u* ile tutulmaktadır.

```
public enum AccEnum
{
    KisaVadeli = 1,
    UzunVadeli = 2,
    Ozel = 3,
    Cari = 4
}
```

➔ CreateAccount Fonksiyonu

Vadeli hesaplardan herhangi biri açılmak istenildiğinde bu fonksiyon çalıştırılır. Parametre olarak *account* nesnesini, *amount* ve *startAmount* değişkenini alır.

```
public void CreateAccount(BankAccount account, string amount, string startAmount)
{
```

Random fonksiyonu ile hesap numarası üretilir.

```
Random rnd = new Random();
int randomNo = rnd.Next(10000000, 99999999);
```

Daha sonrasında kullanıcının hesap türü seçimine göre hesap bilgilerinin sınıflara kayıtları yapılır. Örnek kod olarak kısa vadeli hesap sürecini ele alacağız, diğer hesaplar içinde aynı yöntem izlenilmiştir.

```
if (amount == "1") //Kısa vadeli
{
    if (Int32.Parse(startAmount) >= 5000) //Min tutar kontrolü
    {
        account.ShortDepositAccounts = new ShortDepositAccount();
        account.ShortDepositAccounts.AccountName = AccEnum.KisaVadeli;
        account.ShortDepositAccounts.Balance = Int32.Parse(startAmount);
        account.ShortDepositAccounts.IncomeRatio = 15;
        account.ShortDepositAccounts.MinRequiredMoney = 5000;
        account.ShortDepositAccounts.AccountNumber = randomNo;
```

İlgili sınıfımızdan yeni bir nesne oluşturduktan sonra hesap adını enum olarak tanımlıyoruz. Bakiye için kullanıcıdan alınan başlangıç değerini veriyoruz. Ödev dokümanında belirtilen hesap türüne ait kâr oranı ve gereken asgari miktar değerlerini tanımlıyoruz. Hesap numarası içinde *Random* fonksiyonu ile oluşturduğumuz 8 haneli sayımızı kullanıyoruz.

```
                account.CheckingAccounts.Balance = account.CheckingAccounts.Balance -
Int32.Parse(startAmount); // vadesiz hesap bakiye güncelleme
                checkingInfoBalance = account.CheckingAccounts.Balance;
```

Güncel bakiyenin gösterilmesi için vadeli hesaba aktarılan bakiyenin, cari hesaptan düşülmesi gerekiyor. Bunun için başlangıç değerini, cari hesap bakiyesinden çıkarma işlemini yapıyoruz. Daha sonrasında hesap durumu için *checkingInfoBalance* değerinide güncelliyoruz.

```
accountList.Add($"\nHesap adı & numarası: {account.ShortDepositAccounts.AccountName} -
{account.ShortDepositAccounts.AccountNumber}"); //Hesap listesi
```

Hesap listesine oluşturulan hesaba ilişkin bilgileri ekliyoruz.

```
if (Int32.Parse(startAmount) >= 1000)
{
    drawList.Add(account.ShortDepositAccounts.AccountNumber);
}
```

Hesaba yatırılan tutarlar 1000 TL'den fazlaysa hesap numara çekiliş listesine eklenmektedir.

```
shortInfoName = account.ShortDepositAccounts.AccountName;
shortInfoNumber = account.ShortDepositAccounts.AccountNumber;
shortInfoBalance = account.ShortDepositAccounts.Balance;
```

Hesap durumunda kullanabilmek için hesaba ait bilgileri değişkenlere tanımlıyoruz. Bu aşamadan sonra kullanıcı konsolde "kısa vadeli hesap oluşturuldu" çıktısını almaktadır. Eğer ilk koşulda startAmount hesabın asgari gereken bakiyesi olan 5000 TL'den daha azsa else bloğuna düşeceği için kullanıcı "belirttiğiniz hesap açılış tutarı gereken asgari miktarın altındadır" çıktısını alacaktır. Tüm bu süreçler diğer hesap türleri içinde aynı şekilde devam etmektedir.

➔ DepositMoney Fonksiyonu

Para yatırma işlemleri için çalıştırılan fonksiyondur. Parametre olarak *account* nesnesini ve *amount* değişkenini alır. Kullanıcının girmiş olduğu yatırılacak miktar olan *amount, TryParse* metodu ile int tipine dönüştürme işleminin yapılıp yapılamayacağını kontrol eder. Eğer olumluysa miktarı *money* değişkenine aktarır.

```
public void DepositMoney(BankAccount account, string amount)
{
    var money = 0;
    int.TryParse(amount, out money);
```

Eğer yatırma tutarı 1000 TL'den fazlaysa hesap çekiliş listesine eklenmektedir.

```
if (money >= 1000)
{
    drawList.Add(account.CheckingAccounts.AccountNumber);
}
```

Cari hesabın mevcut bakiyesinin üzerine *money* değerini ekleyerek yatırma işlemini yapıyoruz. Tüm yatırma ve çekim işlemleri cari hesap üzerinden gerçekleşmektedir.

```
account.CheckingAccounts.Balance += money;
checkingInfoBalance = account.CheckingAccounts.Balance;
```

Burada mevcut banka uygulamalarından esinlenilmiştir. Uygulama içerisinde farklı işlemler yapılması için cari hesaba kaynak sağlanması beklenmektedir. Yatırma işlemi tamamlandıktan sonra hesap işlem kayıtları için *historyList* listesine işlem eklenir.

```
 historyList.Add($"\n{account.CheckingAccounts.AccountNumber} numaralı hesaba {money} TL tutarında
yatırma işlemi yapılmıştır.");
}
```

➔ WithdrawMoney Fonksiyonu

Para çekme işlemleri için çalıştırılan fonksiyondur. Parametre olarak *account* nesnesini ve *amount* değişkenini alır. Kullanıcının girmiş olduğu çekilecek miktar olan *amount, TryParse* metodu ile int tipine dönüştürme işleminin yapılıp yapılamayacağını kontrol eder. Eğer olumluysa miktarı *money* değişkenine aktarır. Eğer işlem tutarı 1000 TL'den fazlaysa hesabı çekiliş listesine ekler.

```
        public void WithdrawMoney(BankAccount account, string amount)
        {
            var money = 0;
            int.TryParse(amount, out money);

            if (money >= 1000)
            {
                drawList.Add(account.CheckingAccounts.AccountNumber);
            }
```

Daha sonrasında cari hesap bakiyesinden *money* değerini çıkartır ve cari hesap bakiyesini günceller. Çekme işlemini hesap işlem kayıtları için historyList listesine ekleyerek fonksiyonu sonlandırır.

```
            account.CheckingAccounts.Balance -= money;
            checkingInfoBalance = account.CheckingAccounts.Balance;

            historyList.Add($"\n{account.CheckingAccounts.AccountNumber} numaralı hesaptan {money} TL
tutarında çekim işlemi yapılmıştır.");
```

➔ ShowAccountList Fonksiyonu

Döngü çalıştırarak *accountList* listesi içerisindeki değerleri ekrana bastırır. Bu fonksiyon için foreach döngüsü tercih edilmiştir.

```
        public void ShowAccountList()
        {
            foreach (string index in accountList)
            {
                Console.WriteLine(index);
            }
        }
```

➔ **ProfitAmount Fonksiyonu**

Hesap durumunda vadeli hesapların 30 günlük kâr hesaplarını yapıp, değişkenlere tanımlayan fonksiyondur.

```
        public void ProfitAmount(BankAccount account, int creditTime)
        {
            if (shortInfoBalance > 0)
            {
                shortInfoProfit = Convert.ToInt32(account.ShortDepositAccounts.Balance) *
account.ShortDepositAccounts.IncomeRatio * creditTime / 36500;
            }
            if (longInfoBalance > 0)
            {
                longInfoProfit = Convert.ToInt32(account.LongDepositAccounts.Balance) *
account.LongDepositAccounts.IncomeRatio * creditTime / 36500;
            }
            if (specialInfoBalance > 0)
            {
                specialInfoProfit = Convert.ToInt32(account.SpecialDepositAccounts.Balance) *
account.SpecialDepositAccounts.IncomeRatio * creditTime / 36500;
            }
        }
```

Parametre olarak account nesnesini ve *creditTime* değişkenini alır. *creditTime* 30 olarak belirlenmiş ve vade sayısını temsil etmektedir. Kâr hesaplamaları yaparken şu fonksiyon kullanılmıştır: *anapara * faiz oranı * vade (gün) / 36500*

➔ **ShowAccountInfo Fonksiyonu**

Hesap durumu seçeneğini kullanıldığında çalıştırılan fonksiyondur. Hesaba ilişkin hesap adı, hesap numarası, hesap bakiyesi ve 30 günlük kâr tutarını gösterir.

```
        public void ShowAccountInfo()
        {
            Console.WriteLine($"\n*** Cari Hesap Bilgileri ***\nHesap adı: {checkingInfoName}\nHesap
numarası: {checkingInfoNumber}\nHesap bakiyesi: {checkingInfoBalance} TL");
            Console.WriteLine($"\n*** Kısa Vadeli Hesap Bilgileri ***\nHesap adı: {shortInfoName}\nHesap
numarası: {shortInfoNumber}\nHesap bakiyesi: {shortInfoBalance} TL\n30 günlük kâr tutarı:
{shortInfoProfit} TL");
            Console.WriteLine($"\n*** Uzun Vadeli Hesap Bilgileri ***\nHesap adı: {longInfoName}\nHesap
numarası: {longInfoNumber}\nHesap bakiyesi: {longInfoBalance} TL\n30 günlük kâr tutarı: {longInfoProfit}
TL");
            Console.WriteLine($"\n*** Özel Hesap Bilgileri ***\nHesap adı: {specialInfoName}\nHesap
numarası: {specialInfoNumber}\nHesap bakiyesi: {specialInfoBalance} TL\n30 günlük kâr tutarı:
{specialInfoProfit} TL");
        }
```

➔ TransactionHistory Fonksiyonu

Döngü çalıştırarak *historyList* listesi içerisindeki değerleri ekrana bastırır. Bu fonksiyon için foreach döngüsü tercih edilmiştir.

```
        public void TransactionHistory()
        {
            foreach (string index in historyList)
            {
                Console.WriteLine(index);
            }
        }
```

➔ MakeDraw Fonksiyonu

Ana menüde yer alan çekiliş seçeneği için kullanılan fonksiyondur. ilk olarak *drawListCount* değişkenine, *drawList* listesinin index miktarı saydırılarak tanımlanır.

```
        public void MakeDraw()
        {
            int drawListCount = drawList.Count();
```

Eğer hesap sayısı 0 değilse işlemlerimizi yaptırmaya başlıyoruz. *Random* fonksiyonuyla 0 ile hesap sayısı arasında bir sayı seçiyoruz. *winAccNum* içerisine tanımladığımız bu sayı *drawList* içerisinde hangi elementin index sayısına eşitse o kazanan olmuş olmaktadır.

```
 if (drawListCount != 0)
            {
                Random rnd = new Random();
                int winAccNum = rnd.Next(0, drawListCount);
                var winnerAcc = drawList.ElementAt(winAccNum);

                Console.WriteLine($"\n*** Çekilişimizi kazanan {winnerAcc} numaralı hesabımızı tebrik
ederiz. ***");
                historyList.Add($"\nÇekilişi {winnerAcc} numaralı hesap kazanmıştır.");
                drawList.Clear();
            }
            else
            {
                Console.WriteLine("\nSon çekilişin ardından yeni bir çekiliş için yeterli işlem kaydı
bulunmamaktadır!");
            }
        }
```

Sonuç ekrana yazdırılarak hesap işlem kayıtlarına eklenir. Eğer ilk koşulda hesap sayısı 0 ise konsola yeterli hesap sayısı olmadığı bilgisi yazdırılır.

SONUÇ

Kullanıcıların giriş bölümünde TC Kimlik Numaraları girmesi beklenmektedir. Daha sonrasında TC Kimlik Numaraları üzerinden cari hesapları tanımlanacak ve ana menü seçeneklerine ulaşabileceklerdir.

Hesap açma tercihlerinde vadeli hesap seçenekleri görüntülenmektedir. Hesapların gerekli şartları konsol üzerinde belirtilmiştir. Kullanıcı vadeli hesap açarken cari hesap bakiyesi üzerinden işlem yapacaktır. Bu nedenle hesap açmadan önce ana menüde yer alan para yatırma özelliği ile cari hesabına gereken tutarı yatırmalıdır. Cari hesap bakiyelerine gereken tutarı yatırdıktan sonra diledikleri hesap türünde hesap açabileceklerdir.

Para çekmek için ana menüde yer alan para çekme seçeneği kullanılacaktır. Çekilmek istenen tutar belirtildikten sonra eğer bakiye yeterliyse cari hesabınızdan tutar düşürülecektir.

Hesap listesi ile sistem içerisinde mevcut olan tüm hesap isimleri ve numaraları gözlenebilmektedir.

Hesap durum ile mevcut hesapların isim, numara, bakiye ve 30 günlük kar oranı bilgilerine ulaşılabilmektedir.

Hesap işlem kayıtları ile mevcut hesaptan sistem üzerinde gerçekleştirilen para yatırma, çekme ve hesap açma hareketleri incelenebilmektedir.

Çekiliş ile mevcut hesaplar arasından bir hesabı sistem, rastgele seçebilmektedir. Eğer farklı oturumlar üzerinden çalışabilen bir sistem olursa farklı kullanım alanları doğabilmektedir.

Çıkış yap ile konsol uygulaması sonlandırılır.