Çağla Yağmur İçer - 19360859027

Bursa Teknik Üniversitesi - Bilgisayar Mühendisliği

Nesneye Yönelik Programlama Ödev Raporu

1. Kisi Sınıfı'ndan miras alan Musteri ve BankaPersonel sınıflarını yaptım.

2. BankaPersonel sınıfına temsilcisi olduğu müşterileri saklamak için musteriler isimli bir ArrayList koydum.

```
ArrayList<Musteri> musteriler;
```

3.Musteri sınıfına, hesaplar ve krediKartlari isimli ArrayList'ler koydum.

```
ArrayList<BankaHesap> hesaplar;
ArrayList<KrediKarti> krediKartlari;
```

4.Bütün sınıflar için toString ve get/set metodlarını yazdım.

5.Musteri sınıfı :

5.1. public void hesapEkle metodu :

```
public void hesapEkle(long tcKimlikNo, int musteriNumarasi,
    BankaHesap yeniHesap;
    //Bütün hesaplara 100000 bakiye verdim. Random koyunca s
    switch (hesapTuru) {
        case "Vadeli":
            //long iban, Date hesapAcilisTarih, double topla
            yeniHesap = new VadeliHesap(main.generateRandomN
            break;
        case "Vadesiz":
            //long iban, Date hesapAcilisTarih, double topla
            yeniHesap = new VadesizHesap(main.generateRandom
            break;
        case "Yatirim":
            //long iban, Date hesapAcilisTarih, double topla
            yeniHesap = new YatirimHesap(main.generateRandom
            break;
        default:
            yeniHesap = null;
            break;
    }
    if (yeniHesap != null) {
        hesaplar.add(yeniHesap);
    }
}
```

Metot içerisinde, BankaHesap sınıfından yeni bir hesap oluşturulur. Hesap türüne bağlı olarak farklı hesap nesneleri oluşturulur. Hesap türü, "Vadeli", "Vadesiz" veya "Yatirim" olabilir.

Switch case kullanılarak Hesap türüne uygun nesne oluşturulur.
Eğer hesap türü "Vadeli" ise, VadeliHesap sınıfının constructor metodu kullanılarak yeni bir VadeliHesap nesnesi oluşturulur. Oluşturulan nesne, belirli özelliklere sahip bir vadeli hesabı temsil eder.

Benzer şekilde, hesap türü "Vadesiz" veya "Yatirim" ise, ilgili hesap türüne ait nesneler oluşturulur ve özellikleri belirtilen değerlerle doldurulur.

Eğer hesap türü geçerli bir tür değilse, yeniHesap değişkenine null atanır.

Son olarak, yeniHesap değişkeninin null olmadığı durumlarda, oluşturulan hesap hesaplar listesine eklenir.

5.2. public void hesapSil metodu :

```
public void hesapSil(long tcKimlikNo, int musteriNumarasi, String ad, String soyad) {
    for (BankaHesap hesap : hesaplar) {
        if (getTcKimlikNo() == tcKimlikNo &&
            getMusteriNumarasi() == musteriNumarasi &&
            getAd().equals(ad) &&
            getSoyad().equals(soyad)) {
            if (hesap.getToplamBakiye() > 0) {
                System.out.println("Lütfen öncelikle bakiyenizi başka bir hesaba aktarınız.");
                return;
            } else {
                hesaplar.remove(hesap);
                System.out.println("Hesap başarıyla silindi.");
                return;
            }
        }
    }
    System.out.println("Hesap bulunamadı.");
}
```

BankaHesap tipinde nesneler tutan hesaplar listesindeki her bir hesap için bir döngü oluşturur. Her döngü adımında, mevcut hesap üzerinde bazı koşulları kontrol eder.

---Hesabın TC kimlik numarası, müşteri numarası, adı ve soyadı, parametre olarak verilen değerlerle eşleşiyorsa:

--Hesabın toplam bakiyesi 0'dan büyükse, yani hesapta bakiye bulunuyorsa:

-"Lütfen öncelikle bakiyenizi başka bir hesaba aktarınız." şeklinde bir mesajı ekrana basar ve metottan çıkar.

--Hesabın toplam bakiyesi 0 veya daha küçükse:

-hesaplar listesinden bu hesabı kaldırır.

-"Hesap başarıyla silindi." şeklinde bir mesajı ekrana basar ve metottan çıkar.

---Eğer hiçbir hesap bulunamazsa, yani yukarıdaki koşullar hiçbir hesap için sağlanmazsa:

--"Hesap bulunamadı." mesajını ekrana basar.

5.3. printArrayList metodu (Ben ekledim)

Amaç : Musteri sınıfındaki ArrayList'leri yazdırmak.

```
public static <T> String printArrayList(ArrayList<T> list) {
    StringBuilder sb = new StringBuilder();
    T hesap;
    for (int i = 0; i < list.size(); i++) {
        hesap = list.get(i);
        sb.append(i + 1 + ". Nesne \n").append(hesap.toString()).append(",\n");
    }
    return sb.toString();
}
```

<T> ifadesi, bir metot veya sınıf tanımının başına eklenerek, bu metot veya sınıfın genel (generic) bir yapıya sahip olduğunu belirtir. Bu sayede hem BankaHesap hem KrediKarti tipinde nesneler içeren arraylistler alabilir.
Bu metot, bir ArrayList'i alır ve içeriğini bir String olarak formatlar.
Metot içerisinde, bir StringBuilder nesnesi oluşturulur. Ardından, bir döngü kullanarak list içindeki her öğeyi alır. Her öğe için, StringBuilder nesnesine sırasıyla bir indeks numarası ve öğenin toString metodu sonucunu ekler. Bu işlem, öğelerin alt alta sıralanmış bir şekilde StringBuilder nesnesine eklenmesini sağlar.
Son olarak, StringBuilder nesnesinin içeriği String olarak döndürülür. Sonuç olarak, herhangi bir türdeki ArrayList nesnesinin içeriğini formatlayarak bir String olarak döndürmeyi sağlar.

6. KrediKarti sınıfı :

6.1. KrediKartiEkle metodu:

```
public void KrediKartiEkle(int kartNumarasi, double limit, Musteri musteri, int musteriNumarasi) {
    KrediKarti yeniKrediKarti = new KrediKarti(kartNumarasi, limit, 0.0); //borc baslangicta 0.0;

    if (musteri.getMusteriNumarasi() == musteriNumarasi && yeniKrediKarti != null) {
        musteri.krediKartlari.add(yeniKrediKarti);
    }
}
```

Metot içerisinde, KrediKarti nesnesi oluşturulur ve gerekli bilgileri alır (kart numarası, limit ve başlangıçta borç 0.0). Bu yeni kredi kartı nesnesi, yeniKrediKarti değişkenine atanır. Daha sonra, bir kontrol yapılır. musteri nesnesinin getMusteriNumarasi() metodu ile aldığı müşteri numarası, parametre olarak verilen musteriNumarasi değeri ile eşleşiyor mu diye kontrol edilir. Aynı zamanda yeniKrediKarti değişkeninin null olmadığından da emin olunur. Eğer bu koşullar sağlanıyorsa, yani müşteri numaraları eşleşiyor ve yeniKrediKarti değeri null değilse, yeni kredi kartı musteri nesnesinin krediKartlari adlı ArrayList'ine eklenir.

6.2. KrediKartiSil metodu:

```
public void KrediKartiSil(int kartNumarasi, double limit, double guncelBorc, Musteri musteri) { //musteri
    if (guncelBorc == 0.0) {
        musteri.getKrediKartlari().removeIf(krediKarti -> krediKarti.getKartNumarasi() == kartNumarasi);
        System.out.println("Kart başarıyla silindi.");
    } else {
        System.out.println("Lütfen öncelikle borç ödemesi yapınız.");
    }
}
```

Metot içerisinde, bir kontrol yapılır. Eğer guncelBorc değeri 0.0 ise (yani kredi kartının borcu yoksa), o zaman silme işlemi gerçekleştirilir.
musteri nesnesinin getKrediKartlari() metodu kullanılarak müşteriye ait kredi kartlarına erişilir. Ardından, removeIf metodu kullanılarak krediKarti nesneleri üzerinde bir koşula göre silme işlemi gerçekleştirilir. Koşul, krediKarti nesnesinin getKartNumarasi() metodu ile aldığı kart numarasının, parametre olarak verilen kartNumarasi değeri ile eşleşmesidir.

Silme işlemi tamamlandıktan sonra, "Kart başarıyla silindi." mesajı ekrana yazdırılır.

Eğer guncelBorc değeri 0.0'dan farklı bir değere sahipse (yani kredi kartının borcu varsa), "Lütfen öncelikle borç ödemesi yapınız." mesajı ekrana yazdırılır. Borç ödenmeden kart silinemez.

6.3. KullanilabilirLimit metodu:

```
public double kullanilabilirLimit(int kartNumarasi, double guncelBorc, Musteri musteri) {
    KrediKarti temp = null;
    // Kart numarasına göre ilgili kredi kartının limitine erişim sağlanabilir
    for(int i = 0 ; i < musteri.getKrediKartlari().size() ; i++) {
        if (musteri.getKrediKartlari().get(i).getKartNumarasi() == kartNumarasi) {
            temp = musteri.getKrediKartlari().get(i);
        }
    }
    // temp'in hala boş olup olmadığını kontrol et (eşleşen kartNumarasi bulunamadı)
    if (temp == null) {
        // Eşleşen kartNumarasi'nin bulunmadığı durumu ele alın, örneğin bir istisna atın
        throw new IllegalArgumentException("No matching kartNumarasi found."); // Throw a
    }

    return temp.limit - guncelBorc;
}
```

Metot içerisinde, geçici bir KrediKarti nesnesi olan temp oluşturulur ve başlangıçta null değeri atanır. Bu değişken, ilgili kredi kartını temsil etmek için kullanılacak.

Daha sonra, bir döngü kullanılarak müşterinin kredi kartları listesine bakılır. Her bir kartın getKartNumarasi() metodu kullanılarak kart numarası kontrol edilir. Eğer kart numarası, parametre olarak verilen kartNumarasi değeri ile eşleşiyorsa, temp değişkenine ilgili kredi kartı atanır.

Döngü tamamlandığında, temp değişkeninin hala null olup olmadığı kontrol edilir. Eğer temp değişkeni hala null ise, yani eşleşen kartNumarasi bulunamadıysa, istisna atılır (throw new IllegalArgumentException) ve "No matching kartNumarasi found." şeklinde bir hata mesajı döndürülür.

Eğer temp değişkeni null değilse, yani eşleşen kartNumarasi bulunduysa, temp.limit - guncelBorc ifadesi kullanılarak kullanılabilir limit hesaplanır. Bu hesaplanan değer metottan döndürülür.

6.4. kampanya metodu:

```
public static void kampanya(KrediKarti kart) {
    if (kart.limit < 5000) {
        System.out.println("Kredi teklifi: Kart borcunuz " + kart.guncelBorc + " TL. Daha yüksek limitli bir kredi kartı için kampanyalara göz
    } else if (kart.limit > 10000) {
        System.out.println("Kredi teklifi: " + kart.limit + " TL üzerindeki limitle özel bir kredi teklifi sizin için mevcut!");
    } else {
        System.out.println("Kredi teklifi: Mevcut limitiniz " + kart.limit + " TL. Size uygun kampanyaları değerlendirin.");
    }
}
```

Metot içerisinde, if-else ifadesi kullanılarak kredi kartının limitine göre farklı kredi teklifleri üretilir.

İlk if bloğu, kredi kartının limitinin 5000'den küçük olduğunu kontrol eder. Eğer durum doğruysa, yani limit 5000'den küçükse, guncelBorc değerini yazdırır.

İkinci else if bloğu, kredi kartının limitinin 10000'den büyük olduğunu kontrol eder. Eğer durum doğruysa, yani limit 10000'den büyükse, kartın limit değerini yazdırır ve Kredi teklifi yapar.

Son else bloğu, yukarıdaki koşulların hiçbirinin sağlanmadığı durumu ele alır. Bu durumda, "Kredi teklifi: Mevcut limitiniz [kartın limit değeri] TL. Size uygun kampanyaları değerlendirin." mesajı ekrana yazdırılır.

7. Krediler sınıfı :

7.1. KrediVer metodu :

```
public void KrediVer(int krediID, int musteriNumarasi, int krediMiktar, Musteri musteri) {//mu
    // MusteriNumarasi'na göre müşteriyi bulma
    Musteri temp = musteri;

    if (temp != null) {
        // Müşterinin hesaplarını kontrol etme
        ArrayList<BankaHesap> hesaplar = temp.getHesaplar();
        double toplamBakiye = 0.0;

        for (BankaHesap hesap : hesaplar) {
            toplamBakiye += hesap.getToplamBakiye();
        }

        // Kredi verme işlemi
        double krediLimiti = toplamBakiye * 0.5; // Yıllık kazancın %50'si kadar kredi limiti

        if (krediMiktar <= krediLimiti) {
            // Kredi verilebilir
            System.out.println("Kredi onaylandı. Kredi ID: " + krediID);
        } else {
            // Kredi limiti aşıldı
            System.out.println("Kredi limiti aşıldı. Kredi onaylanmadı.");
        }
    } else {
        // Müşteri bulunamadı
        System.out.println("Müşteri bulunamadı. Kredi onaylanmadı.");
    }
}
```

Metot içerisinde, temp adında bir Musteri nesnesi oluşturulur ve başlangıçta musteri parametresine atanır. Bu değişken, kredi verilecek müşteriyi temsil etmek için kullanılır.

Sonra, temp değişkeninin null olmadığı kontrol edilir. Eğer temp değişkeni null değilse, yani müşteri bulunduysa, müşterinin hesaplarını kontrol etmek için hesaplar adında bir ArrayList<BankaHesap> oluşturulur ve müşterinin hesaplarını temp.getHesaplar() metodu ile alır.

Daha sonra, tüm hesaplardaki toplam bakiye hesaplanır. Bunun için bir döngü kullanılır ve her bir hesabın bakiyesi hesap.getToplamBakiye() metodu ile alınarak toplamBakiye değişkenine eklenir.

Sonraki adımda, kredi limiti hesaplanır. Toplam bakiyenin %50'si kadar kredi limiti belirlenir ve krediLimiti değişkenine atanır.

Şimdi, krediMiktar parametresi, hesaplanan kredi limiti ile karşılaştırılır. Eğer krediMiktar kredi limitinden küçük veya eşitse, kredi verilebilir ve "Kredi onaylandı. Kredi ID: [krediID]" mesajı ekrana yazdırılır.

Eğer krediMiktar kredi limitini aşıyorsa, kredi limiti aşıldığı için "Kredi limiti aşıldı. Kredi onaylanmadı." mesajı ekrana yazdırılır.

Eğer temp değişkeni null ise, yani müşteri bulunamadıysa, "Müşteri bulunamadı. Kredi onaylanmadı." mesajı ekrana yazdırılır.

8.VadeliHesap sınıfı :

Constructor, get/set ve toString metodları var. Açıklanacak ek bir metod yok.

9.VadesizHesap sınıfı :

9.1. paraTransfer metodu : (çok uzun resim yapıştırmayacağım. bu sınıftaki ilk metod)

Bu metot, hesap türü ve hesap bilgisi parametrelerine göre kesinti ve faiz oranı hesaplayarak para transferinin gerçekleştirileceği hesaplamaları yapar ve sonuç olarak transfer miktarını döndürür.

Metot içerisinde, kesinti, faizOrani ve iban değişkenleri başlangıçta sıfır olarak tanımlanır. iban değeri, üst sınıf olan super üzerinden getIban() metodu ile alınır.

Daha sonra, hesapTuru parametresi "Vadeli" ise ve hesapBilgisi 1 ise (maaş hesabı), faiz oranı 0.2 ve kesinti 0.0 olarak atanır. hesapBilgisi 2 ise (normal hesap), faiz oranı 0.1 ve kesinti 8.0 olarak atanır.

Eğer hesapTuru parametresi "Vadesiz" ise, faiz oranı ve kesinti değerleri hesapBilgisi'ne göre atanır. Maaş hesabı için faiz oranı ve kesinti değeri 0.0 olarak atanır. Normal hesap için faiz oranı 0.0 ve kesinti 8.0 olarak atanır.

Eğer hesapTuru parametresi "Yatirim" ise, faiz oranı ve kesinti değerleri 0.0 olarak atanır.

Daha sonra, hesap bakiyesi vadesizBakiye ve kesinti değerleri kullanılarak hesaplanır.

Faiz miktarı, hesap bakiyesi ile faiz oranının çarpımı olarak hesaplanır.

Transfer miktarı, işlem miktarından faiz miktarının çıkarılmasıyla hesaplanır.

Son olarak, transferMiktari değişkeni döndürülür.

9.2. krediKartiBorcOdeme metodu :

```
public void krediKartiBorcOdeme(double odemeMiktari, KrediKarti krediKarti)
    // Hesabın bakiyesinden ödeme miktarı kadar düşürme
    double yeniBakiye = vadesizBakiye - odemeMiktari;
    setVadesizBakiye(yeniBakiye);

    // Kredi kartının güncel borcunu ödeme miktarı kadar azaltma
    double guncelBorc = krediKarti.getGuncelBorc();
    double yeniBorc = guncelBorc - odemeMiktari;
    krediKarti.setGuncelBorc(yeniBorc);
}
```

Metodun içinde, hesabın bakiyesinden ödeme miktarı kadar düşürme işlemi gerçekleştirilir. Bu işlem, mevcut bakiyeden odemeMiktari kadar çıkartma yaparak yeni bakiyeyi hesaplar. Bu değer, yeniBakiye değişkenine atanır.

setVadesizBakiye(yeniBakiye) çağrısıyla, hesabın vadesiz bakiyesi yeniBakiye değeriyle güncellenir.

Ardından, kredi kartının güncel borcunu ödeme miktarı kadar azaltma işlemi gerçekleştirilir. Mevcut borç değeri guncelBorc değişkenine atanır.

yeniBorc değişkenine, mevcut borçtan odemeMiktari kadar çıkartma yaparak yeni bir borç değeri hesaplanır.

krediKarti.setGuncelBorc(yeniBorc) çağrısıyla, kredi kartının güncel borcu yeniBorc değeriyle güncellenir. Bu, guncelBorc adında bir alanın değerini değiştirme işlemidir.

9.3. krediOdeme metodu :

```
public double krediOdeme(int krediID, Krediler krediMiktari, Krediler taksitMiktari) {
    // Hesabın bakiyesinden ödeme miktarı kadar düşürme
    double odemeMiktari = taksitMiktari.getTaksitMiktar();
    double yeniBakiye = getVadesizBakiye() - odemeMiktari;
    setVadesizBakiye(yeniBakiye);

    // Kredinin güncel borcunu ödeme miktarı kadar azaltma
    double krediBorc = krediMiktari.getKrediMiktari();
    double yeniBorc = krediBorc - odemeMiktari;
    krediMiktari.setKrediMiktari(yeniBorc);

    // Ödenen miktarı döndürme
    return odemeMiktari;
}
```

Metodun içinde, ödeme miktarı taksitMiktari'ndan alınır. Bu, taksitMiktari nesnesinin getTaksitMiktar() metodunu çağırarak elde edilir. Ödeme miktarı odemeMiktari değişkenine atanır.

Hesabın bakiyesinden ödeme miktarı kadar düşürme işlemi gerçekleştirilir. Mevcut vadesiz bakiye değerinden odemeMiktari çıkartılarak yeni bir bakiye değeri hesaplanır. Bu değer yeniBakiye değişkenine atanır.

setVadesizBakiye(yeniBakiye) çağrısıyla, hesabın vadesiz bakiyesi yeniBakiye değeriyle güncellenir. Bu, vadesizBakiye adında bir alanın değerini değiştirme işlemidir.

Kredinin güncel borcunu ödeme miktarı kadar azaltma işlemi gerçekleştirilir. Mevcut kredi miktarı krediMiktari'nın getKrediMiktari() metodunu çağırarak elde edilir ve krediBorc değişkenine atanır.

yeniBorc değişkenine, mevcut kredi borcundan odemeMiktari kadar çıkartma yaparak yeni bir borç değeri hesaplanır.

krediMiktari.setKrediMiktari(yeniBorc) çağrısıyla, kredinin güncel borcu yeniBorc değeriyle güncellenir. Bu, krediMiktari adında bir alanın değerini değiştirme işlemidir.

Son olarak, ödenen miktar olan odemeMiktari değeri return edilir.

10. YatirimHesap sınıfı :

10.1. paraEkle metodu :

```
public double paraEkle(int yatirimBakiye, String yatirimTuru, float kur, int yatirilanMiktar) { //yatirilanMiktar ekledim
    double eklenecekMiktar = yatirilanMiktar * kur;
    this.yatirimBakiye += eklenecekMiktar;
    this.yatirimTuru = yatirimTuru;
    System.out.println("Hesaba " + eklenecekMiktar + " birim " + yatirimTuru + " eklendi. Son Bakiye : " + yatirimBakiye);
    return this.yatirimBakiye;
}
```

Metodun içinde, eklenecek miktar hesaplanır. yatirilanMiktar ile kur çarpılır ve sonuç eklenecekMiktar değişkenine atanır. Bu işlem, yatırılan miktarın belirtilen kur ile çarpılmasıyla hesaplanan toplam miktarı temsil eder.

this.yatirimBakiye += eklenecekMiktar ifadesiyle, hesap bakiyesine eklenecekMiktar kadar eklemeyi gerçekleştirir. Bu, yatirimBakiye adında bir alanın değerini artırma işlemidir.

this.yatirimTuru = yatirimTuru ifadesiyle, yatırım türü alanı (yatirimTuru) yeni bir değerle güncellenir.

"Hesaba " + eklenecekMiktar + " birim " + yatirimTuru + " eklendi. Son Bakiye : " + yatirimBakiye ifadesi, eklendiğinde konsola yazdırılan bir mesajdır. Bu mesaj, eklenen miktarın türü ve hesabın güncellenmiş bakiyesini içerir.

Son olarak, güncellenmiş bakiye olan this.yatirimBakiye değeri return edilir.

10.2. paraBoz metodu :

```
public double paraBoz(int yatirimBakiye, String yatirimTuru, float kur) {
    double cekilecekMiktar = yatirimBakiye * kur;
    if (cekilecekMiktar <= this.yatirimBakiye) {
        this.yatirimBakiye -= cekilecekMiktar;
        System.out.println("Hesaptan " + cekilecekMiktar + " birim " + yatirimTuru + " çekildi.");
    } else {
        System.out.println("Hesapta yeterli " + yatirimTuru + " bulunmamaktadır.");
    }
    return this.yatirimBakiye;
}
```

Metodun içinde, çekilecek miktar hesaplanır. yatirimBakiye ile kur çarpılır ve sonuç cekilecekMiktar değişkenine atanır. Bu işlem, çekilecek miktarın belirtilen kur ile çarpılmasıyla hesaplanan toplam miktarı temsil eder.

cekilecekMiktar <= this.yatirimBakiye ifadesiyle, çekilecek miktarın hesap bakiyesinden küçük veya eşit olup olmadığı kontrol edilir. Eğer çekilecek miktar hesap bakiyesinden küçük veya eşitse, içeriye girilir.

this.yatirimBakiye -= cekilecekMiktar ifadesiyle, hesap bakiyesinden cekilecekMiktar kadar düşürme işlemi gerçekleştirilir. Bu, yatirimBakiye adında bir alanın değerini azaltma işlemidir.

"Hesaptan " + cekilecekMiktar + " birim " + yatirimTuru + " çekildi." ifadesi, çekilen miktarı ve çekildiği yatırım türünü konsola yazdıran bir mesajdır.

Eğer çekilecek miktar hesap bakiyesinden daha büyük ise, else bloğuna girilir.

"Hesapta yeterli " + yatirimTuru + " bulunmamaktadır." ifadesi, yeterli miktarda yatırımın hesapta bulunmadığını belirten bir mesajı konsola yazdırır.

Son olarak, güncellenmiş bakiye olan this.yatirimBakiye değeri return edilir.

11. main sınıfı :

İlk olarak, Musteri sınıfından musteri1 ve musteri2 nesneleri oluşturulur.
Bu nesnelerin özellikleri rastgele değerlerle doldurulur.

Ardından, KrediKarti sınıfından kartEkleyici nesnesi oluşturulur ve rastgele değerlerle doldurulur.

kartEkleyici nesnesi kullanılarak KrediKartiEkle metodu çağrılır ve her bir müşteri için bir kredi kartı oluşturulur.

musteri1 ve musteri2 nesnelerinden ilgili kredi kartları kart1 ve kart2 değişkenlerine atanır.

musteri1 ve musteri2 nesnelerine vadesiz ve vadeli hesaplar eklenir.

Kullanıcıdan bir müşteri numarası alınır.

Girilen müşteri numarasına göre, musteri1 veya musteri2 nesnelerinin bilgileri ekrana yazdırılır. (musteri1 için 1 musteri2 için 2 girilmeli.)

kart1 ve kart2 nesneleri için kampanya metodu çağrılarak kredi kartı limiti kontrol edilir ve uygun mesajlar yazdırılır.

musteri1 ve musteri2 nesnelerinden vadesiz ve vadeli hesaplar VadesizHesap ve VadeliHesap türlerine dönüştürülür.

kullanilabilirLimit metodu kullanılarak musteri1 ve musteri2 için kullanılabilir kredi kartı limitleri hesaplanır ve yazdırılır.

hesap1 nesnesinin paraTransfer metodu çağrılarak para transferi gerçekleştirilir.

toplamBakiyeYazdir metodu kullanılarak müşterilerin hesaplarının toplam bakiyeleri hesaplanır ve yazdırılır.

Programın çalışması sona erer.

Ayrıca, toplamBakiyeYazdir, tcKimlikNoRandom, telNoRandom ve generateRandomNumber gibi yardımcı metotlar da bulunmaktadır. Bu metotlar, rastgele sayılar ve kimlik numaraları oluşturmak gibi belirli işlevleri yerine getirir.