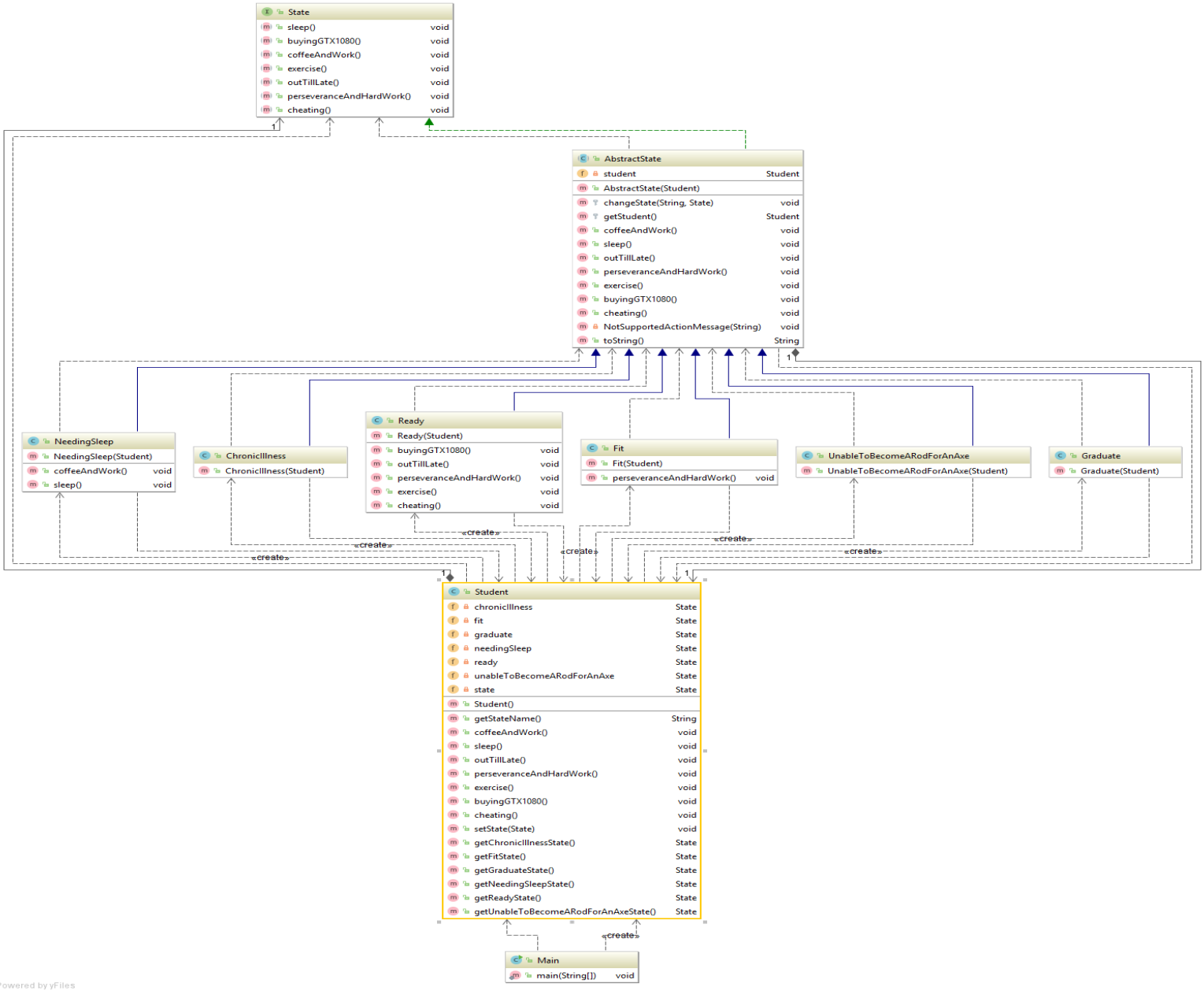
CSE443 Object Oriented Analysis and Design

Seyit Ahmet KARACA - 141044084

HW4 Rapor

1.Part 1 – State Design Pattern

1.1.Part1 UML Diyagramı



1.2.UML Diagram Açıklaması

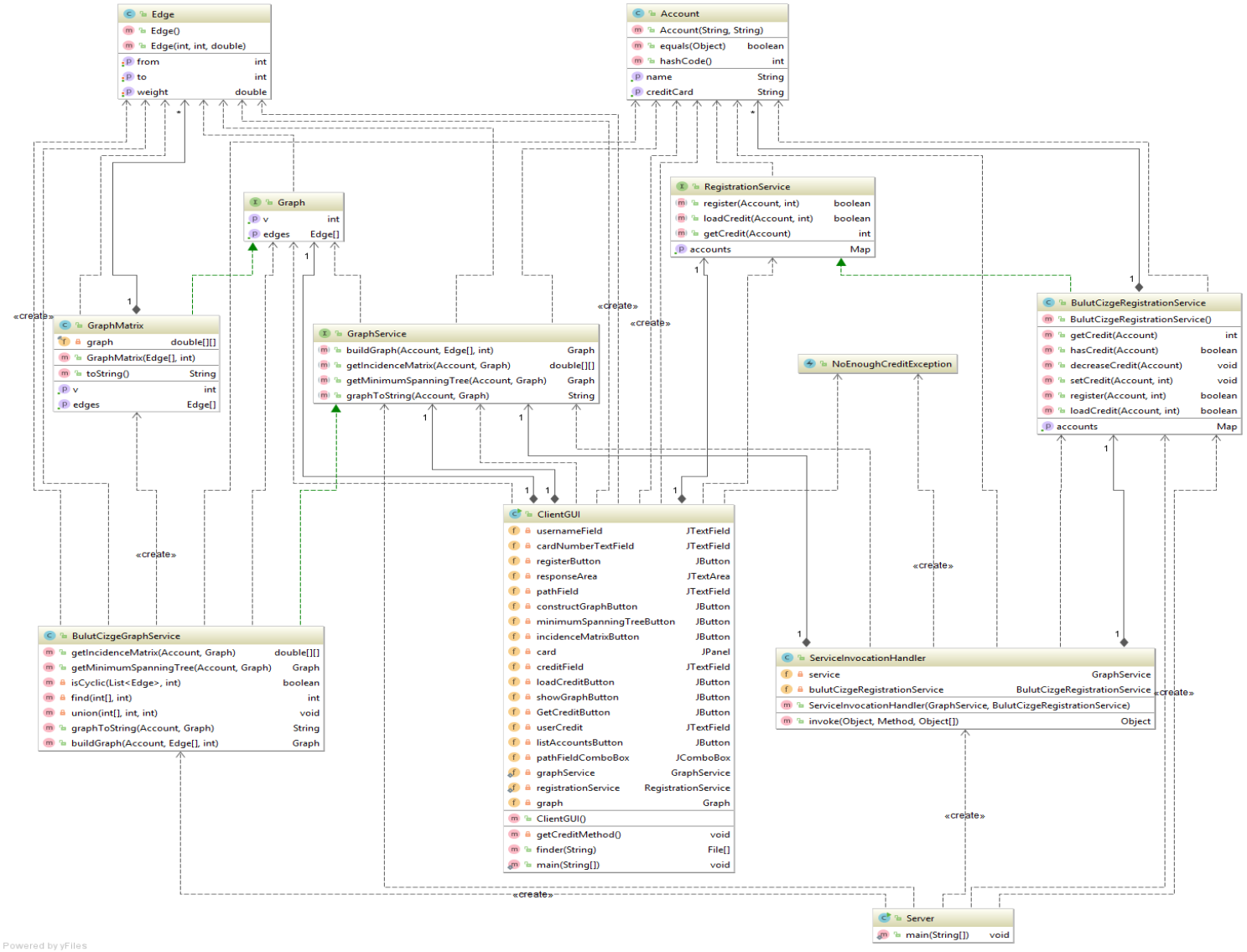
State'lerin bulunduğu bir interface oluşturuldu.Bu state'leri implement edip hepsine hata mesajı bastırılan ve stateleri değiştirilmesini sağlayan bir soyut sınıf oluşturuldu. Soyut sınıf içerisinde bir tane öğrenci sınıfı değişkeni tutulup bu öğrenci sınıfında ise öğrencinin yapabileceği aktiviteler tanımlandı ve aktivitelere göre state'i güncellendi.Her durum için soyut sınıftan türetilmiş şekilde sınıf oluşturuldu ve bu durum sınıflarından hangi durumlara geçilebiliyor ise ilgili durumlar override edildi. Her durum sınıfında bulunduğu durumdan geçebileceği duruma ait olan method çağrıldığında durum güncellemesi yapıldı.

1.3.State Design Pattern Sonuçları

```
---ready to unable to become a rod for an axe-----------------------------------
State ChronicIllness is not supporting action buying GTX1080
Current state : Ready + Cheating
New state : UnableToBecomeARodForAnAxe

---ready to needing sleep to ready----------------------------------------------
Current state : Ready + Out till late
New state : NeedingSleep

Current state : NeedingSleep + SLEEP
New state : Ready

---Errors-----------------------------------------------------------------------
State Ready is not supporting action coffee & work
Current state : Ready + Out till late
New state : NeedingSleep

State NeedingSleep is not supporting action cheating
State NeedingSleep is not supporting action perseverance and hard work
State NeedingSleep is not supporting action buying GTX1080
State NeedingSleep is not supporting action exercise

---ready to fit to graduate---------------------------------
Current state : Ready + Exercise
New state : Fit

Current state : Fit + Perseverance and hard work
New state : Graduate

---ready to graduate----------------------------------------
Current state : Ready + Perseverance and & work
New state : Graduate

---ready to needing sleep -> chronic illness----------------
Current state : Ready + Out till late
New state : NeedingSleep

Current state : NeedingSleep + Coffee & work
New state : ChronicIllness
```

2. Part2 - Proxy Design Pattern

2.1. Part2 - Proxy UML Diagramı



Server-Client-Proxy UML Diagramı

2.2. UML Diagram Açıklaması

Öncelikle hem server hem de client tarafında kullanılacak sınıf ve interface'ler oluşturuldu. Kullanıcıların kullanıcı adı ve kredi kartı bilgilerini tutan Account , Graph için kullanılacak Edge sınıfı , Graph interface'i , client tarafından yapılacak istekler için RemoteException fırlatan GraphService interface'i , bakiye yetersizliğini belirten NoEnoughCreditException sınıfı ve server'a kayıt olabilmek için içerisinde RemoteException fırlatan methodların bulunduğu RegistrationService interface'idir.

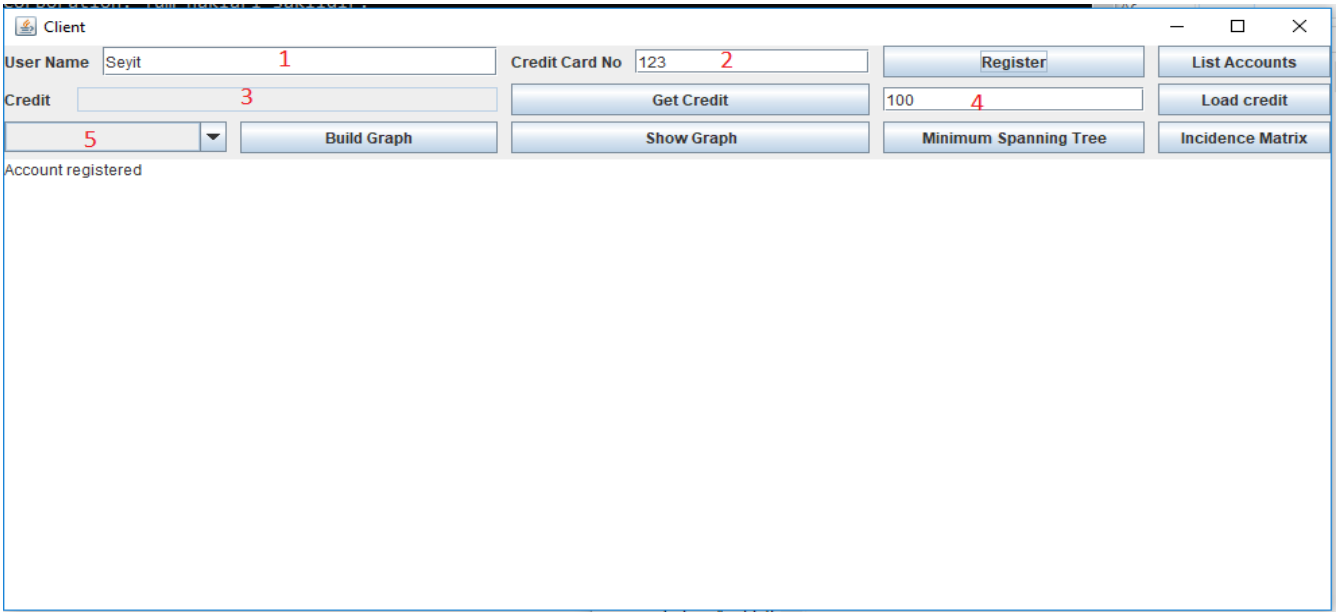
Server tarafında hizmet edecek GraphService'i implement eden BulutCizgeGraphService sınfı oluşturuldu. Bu sınıf GraphSerive'deki methodları implement etmektedir.Minimum spanning tree gibi hizmetler bu sınıf üzerinden gerçekleştirilecektir. RegistrationService'i implement eden BulutCizgeRegistrationService sınıfı ise kullanıcıların kayıt olabilmesi hizmetini gerçekleştirecektir. Kullanıcılar ve kredi bilgileri bu sınıf üzerinden işleme alınıyor.Client tarafından gelen istekleri karşılayacak invocationHandler interfaceini implement etmiş ServiceInvocationHandler bulunmaktadır. Client ile haberleşmede arada bulunan sınıftır. Server tarafında son olarak proxy'ye kayıt edilecek servislerin tanımı yapılan , graph service ve registration servisin kayıt edildiği , rmiregistry'nin açıldığı Server sınıfı bulunmaktadır.

Client tarafında ise grafik arayüzü bulunmaktadır. Grafik arayüzü başlatılmadan önce Naming.lookup ile servisler tanıtılmış ve gerekli butonlara servislerle haberleşmesi için kodlar yazılmıştır.

2.3. Part2 – Proxy Design pattern Çıktıları ve notlar

Programın sağlık çalışabilmesi için client.jar ile aynı klasörde bulunması gereken graph'ların bulunduğu text dosyaları bulunmalıdır. Text dosyaların içeriği ilk satırda vertex sayısı , alt satırlarda ise from(int) to(int) weight(int) şeklinde satırlardan oluşması gerekmektedir.

Öncelikle komut satırından server.jar bulunduğu klasöre gidilip " java –jar server.jar" çalıştırılmalıdır. Sonrasında "java –jar client.jar" ile client GUI açılmalıdır. Server başlatılmadan client GUI açılmak istenir ise hata ile mesajı bastırılıp client.jar kapanmaktadır.



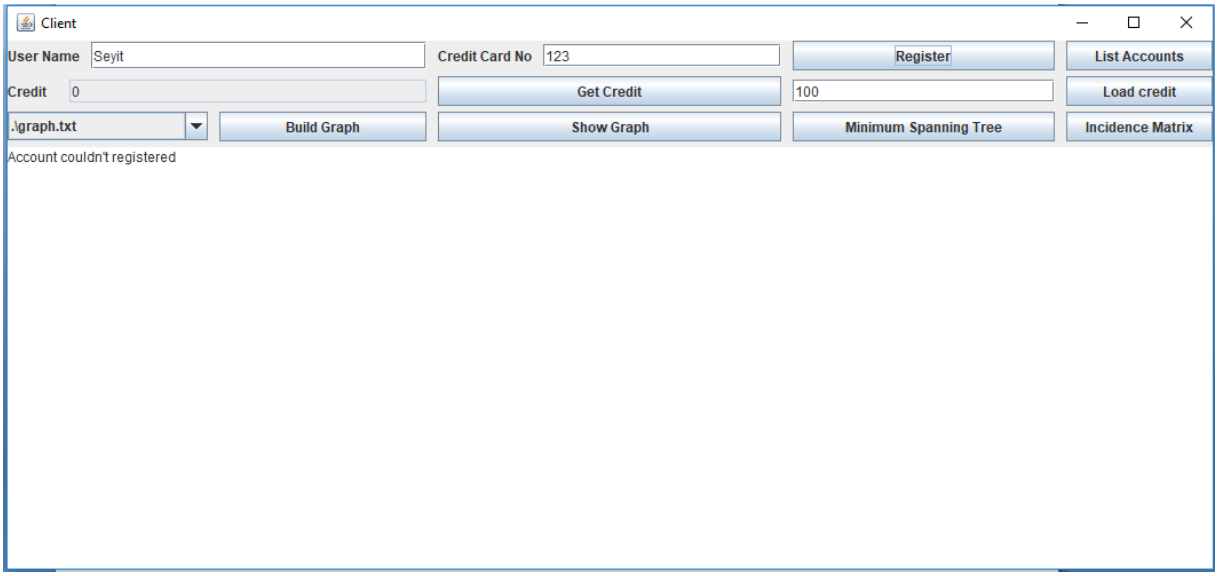
İlk başlatıldığında kullanıcı grafik arayüzü

Kırmızı 1 rakamı ile işaretlenmiş alana kullanıcı adı girilir. Kırmızı 2 rakamı ile işaretlenmiş alan kredi kartı numarası için gereklidir. Bunlar doldurulduktan sonra "Register" butonuna tıklanır.

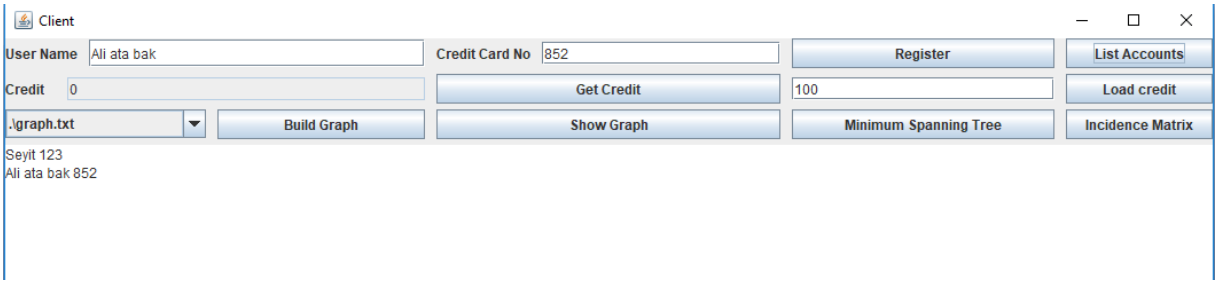
3 numaralı alan "Get Credit" butonuna veya herhangi bir servis kullanıldığında kalan bakiyeyi göstermektedir. 4 numaraları alanda bulunan sayı "Load Credit" butonuna basılarak kalan bakiyeye 4

numaralı alanda yazan miktar kadar yükleme yapılır. 5 ile işaretlenmiş alanın kullanım şekli ise , aşağı ok işaretinden önce 5 numaralı alana bir kez tıklanarak client.jar ile aynı klasörde bulunan text dosyalarını tarar ve bu combobox doldurulur. Sağındaki ok'a basılarak istenen içerisinde graph olan dosya seçilip build graph , show graph , minimum spanning tree , incidence matrix servisleri kullanılabilir. Her servis 2 kredi ile kullanılmaktadır ve başlangıçta her kullanıcıya 20 kredi verilmektedir. List Account servisi ise servera kayıtlı kullanıcıları alttaki panelde listelemektedir ve kullanıcıya bağlı bir servis olmadığından herhangi bir kredi ücretine tabii tutulmamıştır.

Çıktılar:

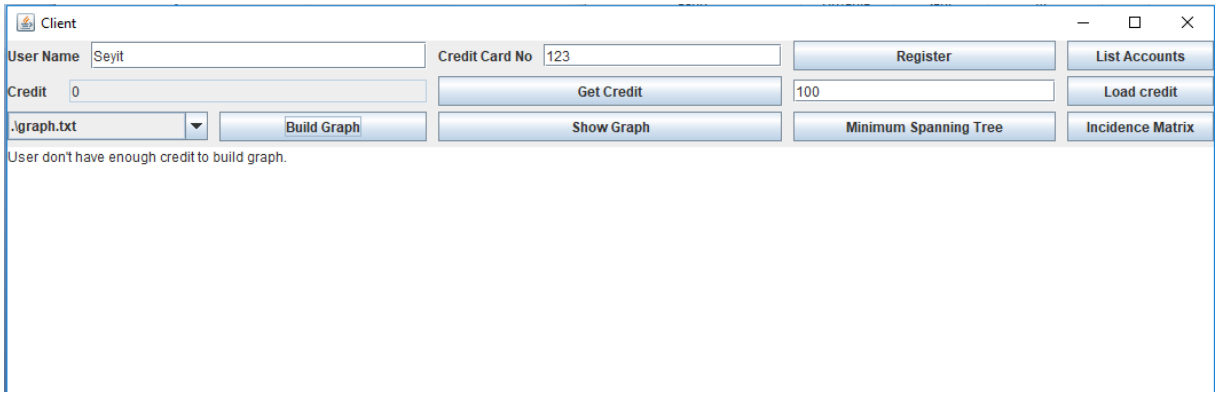


Sisteme kayıt olan bir kullanıcı tekrar kayıt olamaz mesajının bulunduğu görsel



.

Yetersiz bakiye bilgisinin bulunduğu görsel.



Sistemdeki kayıtlı kullanıcıların bulunduğu görsel.

```
graphToString invoked at 07:29:02 (Account: qwe) (Credit left: 18) (Thread 19)
buildGraph invoked at 07:29:17 (Account: qwe) (Credit left: 16) (Thread 19)
buildGraph tooks 0,011 ms (Account: qwe) (Thread 19)
 ------------------------------


graphToString invoked at 07:29:19 (Account: qwe) (Credit left: 14) (Thread 19)
graphToString tooks 0,326 ms (Account: qwe) (Thread 19)
 ------------------------------


getMinimumSpanningTree invoked at 07:29:21 (Account: qwe) (Credit left: 12) (Thread 19)
getMinimumSpanningTree tooks 0,944 ms (Account: qwe) (Thread 19)
 ------------------------------
```

Server'da hangi servisin hangi kullanıcı tarafından saat kaçta kullanıldığı ve ne kadar sürdüğü bilgisinin çıktısıdır.

Not :Derste gösterdiğiniz konsoldan "start rmiregistry" komutu server sınıfı içerisine gömülmüştür. Bu yüzden kontroldan "start rmiregister" komutu yapılmasına gerek yoktur. Executable dosyaları ödev klasörü içerisinde executablePart1 ve executablePart2 dosyalarında bulunmaktadır. Görsellerin orjinalleri ödev klasöründe bulunmaktadır. odevPart1 ve odevPart2 ' de ise proje kodları bulunmaktadır.

Not2: Bu ödevin kodlarında bazı yerlerde odev5 görebilirsiniz. Ödeve başladığımda bu ödevi 5.ödev sanıyordum o yüzden öyle bir yanlışlık yaptım. Kodlar düzgün çalışıyor, sadece bir isim karışıklığı oluştu.

Not 3 : Çalıştığım makinedeki java –version çıktısı;

java version "1.8.0_191"

Java(TM) SE Runtime Environment (build 1.8.0_191-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)