



OBJECT ORIENTED ANALYSIS AND DESIGN

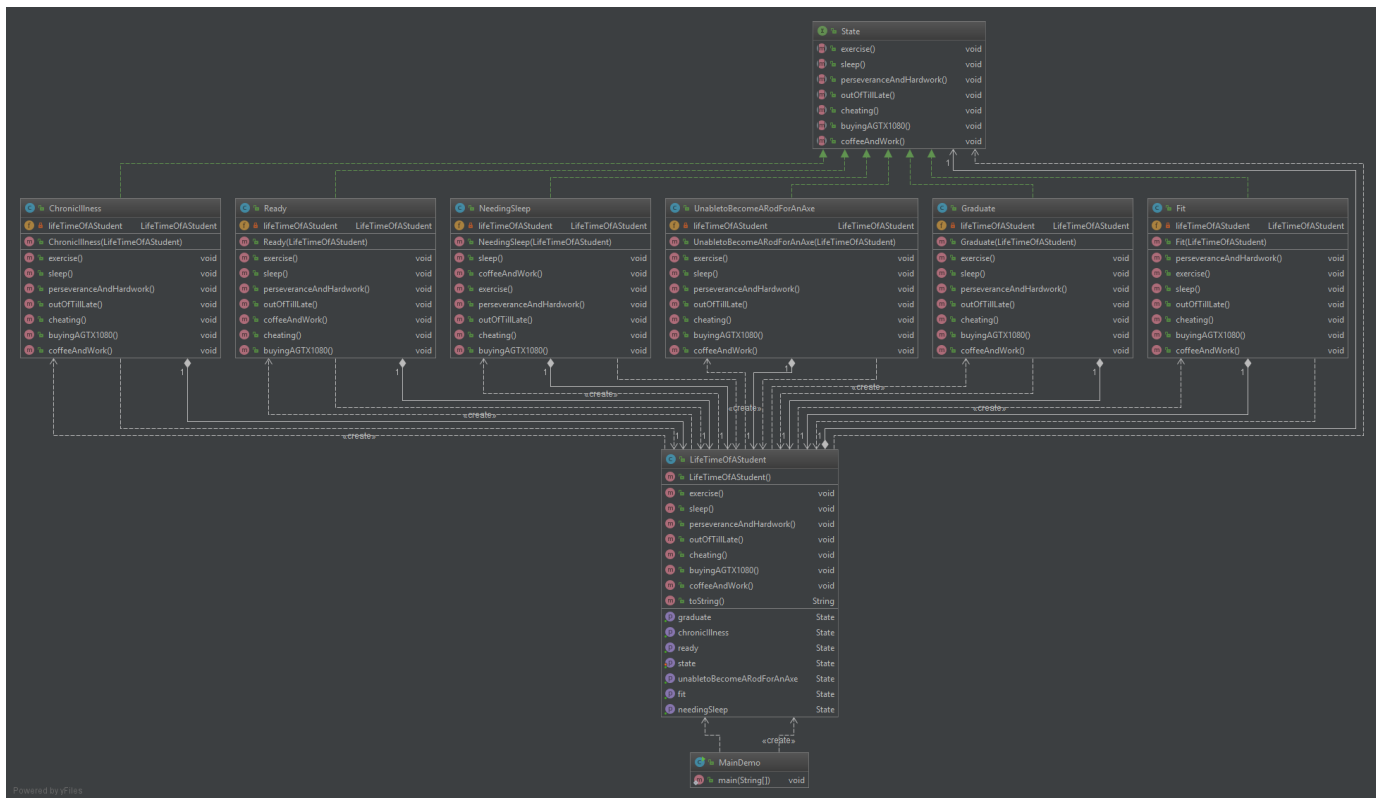
Homework 4

Fürkan YILDIZ
141044031

Part1:

Öncelikle State interfacesi oluşturularak, finite state machine’deki tüm aktiviteler bu interfacede method olarak yazıldı. Daha sonra her state için kendi adında (örneğin “Fit”) class oluşturularak, bu methodlar o state için implement edildi. (Örneğin “Fit” statesinde “perserverance & hard work” aktivitesi gerçekleştirilebilir sadece, gerçekleştirildiğinde ise “Graduate” state’sine geçiş yapılır. Dolayısıyla Fit sınıfındaki “perserverance & hard work” methodu çağırıldığında, studentin statesi “Graduate” olarak değiştiriliyor. Diğer methodlar ise Fit statesinde çağırılmayacaklarından “UnsupportedOperationException” fırlatılıyor). State’ler bağlı oldukları machine’nin (LifeTimeOfAStudent) state’sini değiştirebileceğinden contructure’lerinde machine’yi alıyorlar.

Machine'nin ana yönetim merkezi ise LifeTimeOfAStudent sınıfı. Bu sınıfta olabilecek tüm Stateler oluşturuluyor ve Student'in state'sine başlangıç statesi olan "Ready" statesi atanıyor. Finate State Machine'de gerçekleştirilebilecek aktiviteler içinse(Exercise,Sleep,...) methodlar yazılıyor ve bu methodlar Current State'te delege ediliyor. Böylece herhangi aktivite çağırıldığında, Student'in o anlı State'si ne ise, o State'nin istenilen methodu çağırılıyor.



Part2:

Öncelikle Graph yazabilmek için “IGraph” interfacesi oluşturuldu bu interfacede add,remove,print ve get methodları eklendi. Add ve remove metodu graph’a edge ekleme çıkartma işlemlerini yapıyor. Print metodu graph’ı print ediyor, get metodu ise graph’ın edgelerini return ediyor. Daha sonra bu interfaceyi implement edebilmek için “Graph” sınıfı yazıldı. Bu sınıf ile herhangi tipte graph oluşturulabileceği için Object tipinde objeler tutuldu.

Graph’da vertex ve edge bilgileri map yardımıyla tutuldu. Her Weight (Double), bir edge’ye(2 vertex) e karşılık geliyor, ve aynı weight’ten birden fazla edge olabileceğinden dolayı her weight için bir edge map listesi tutuldu. Yani graph’ımızın “Map<Double,List<Map<Object,Object>>>” şeklinde saklanıyor. Double Weight için, 2 adet Object ise Vertex’ler için tutuluyor.

Daha sonra remove servisenin methodlarını belirtmek için java.rmi’ın “Remote” sınıfını extend eden “IRemoteService” interfacesi oluşturuldu ve bu interfaceye remote servisin iki methodu olan minimumspanning tree ile incidence matrix methodları koyuldu. Daha sonra bu methodları implement edebilmek için “RemoteService” sınıfı oluşturuldu, bu sınıf interfacesini implement etmenin yanı sıra, rmi bağlantısını kurabilmek için “UnicastRemoteObject” sınıfını extend ediyor ve constructuresinde, bu extend ettiği sınıfın constructuresini çağırıyor.

Minimum Spanning Tree’yi (Kruskal Algorithm ile) implement ederken weightleri sıralamamız gerektiğinden, bu zahmetten kaçınmak için, Graph’ı tuttuğumuz Map’i “TreeMap” ile implement ettim. Böylece Graph iterate edilirken, Weightlere göre küçükten büyüğe giden şekilde iterate edilecek.

“minimumSpanningTree” methodu rmi ile client tarafından oluşturulan IGraph’ ı parametre olarak alarak bu graph’ın minimum spanning treesini hesaplamayı amaçlamaktadır.

“minimumSpanningTree” methodunda, Kruskal Algoritmasına göre minimum spanning tree hesaplandı. Bu algoritmaya göre edgelerin weightleri küçükten büyüğe sıralanır(Tree map kullanıldığından zaten map yapısında sıralı olarak saklanıyor graph). Daha sonra bu edgeler sırasıyla gezilir ve cycle oluşturmayacak edgeler (iki vertex’inden en az birisi daha önce gezilmemiş edgeler) minimum spanning tree’ye dahil edilir. Edgeler gezilirken, bu methodun içerisinde oluşturulan minimum spanning tree IGraph’ına bahsedilen koşulu sağlayan edgeler eklenir. Tüm edgeler iterate edildikten sonra ise minimum spanning tree IGraph’ı return edilir.

“IncidenceMatrix” methodunda, Graph’ın Incidence Matrix’ini hesaplamak için önce vertex ve edge sayısı hesaplandı daha sonra her vertex için edge sayısı kadar 0 (sıfır) içeren bir liste tutuldu. Böylece yapı “Map<Object,int[]>” haline geldi. Daha sonra tüm edgeler gezilerek, vertex lerde tutulan liste baştan sonra edgenin çıktığı vertex için 1, girdiği vertex için -1 olacak şekilde listeler update edildi. Böylece her edge için source vertexi 1, destination vertexi -1 ve o edge ile alakası olmayan diğer vertexler 0 olacak şekilde incidence matrixi oluşturuldu.

Client’te GUI açabilmek için GUI Sınıfı yazıldı. GUI açıldığında remote servis’e otomatik olarak bağlanmaya çalışıyor (constructuresi ile lookup yapıyor). GUI’ de graph’ın edgeleri manuel olarak girilerek, clientte graph oluşturulabiliyor. Graph’ı oluşturduktan sonra ise remote servis ile gerçekleştirebileceği fonksiyonları (minimum spanning tree, incidence matrix) gerçekleştirebilmesi için bu servisler buton olarak eklendi. GUI açıldığında bağlanma gerçekleşmezse remote servis fonksiyonlarını çalıştıramıyor normal olarak ve “servera bağlanılmadı” şeklinde bir uyarı veriyor. Bağlandığında ise remote service butonlarına tıklandığında remote servisin stub’ına graph objesi ve gerekli diğer parametreler(clientID) verilerek, graph server’a rmi ile gönderiliyor, işlemler serverda yapılıyor ve rmi ile sonuç cliente geliyor ve sonuç GUI’de bastırılıyor. Minimum spanning tree için,

minimum spanning tree'nin graph'ı , incidence matrix içinse incidence matrix (matrix formatında) bastırıldı.

1.4 de istenilen credit hesaplaması için, Server'da clientlerin ID-credit ilişkisinin tutulması gerekir. Her client'in ödediği para miktarına göre ona credit verebilmek için. Bunu yapabilmek için serverda basitçe integer array'i tutuldu. Arrayin her elemanında, bir client için credit bilgisi tutuluyor. Client id leri ise 0 dan başlıyor ve her abone client için 0,1,2,3,4,... şeklinde devam ediyor. Şuan server'ıma 10 size'ı olan array tutarak, 10 adet client kaydetmiş oldum, client id leri 0,1,2,...,9 şeklinde. Bu istenildiği kadar arttırılabilir.

Server'ın kendisini tanıması için her client'in ise kendi id'si olmalı ve remote serviceleri çağırırken bu id yi de parametre ile göndermeli servera. Böylece server, client'i tanıyacak, yaptığı her işlem için credit'ini azaltacak ve eğer credit'i istediği işlem için yetersiz ise exception fırlatacaktır. (GUI'de bu exception yakalanarak ekrana uyarı olarak basılmaktadır.)

Thread-safe içinse serviceler "synchronized" yapıldı.

Client creditleri ve costlar şu şekilde ayarlandı,

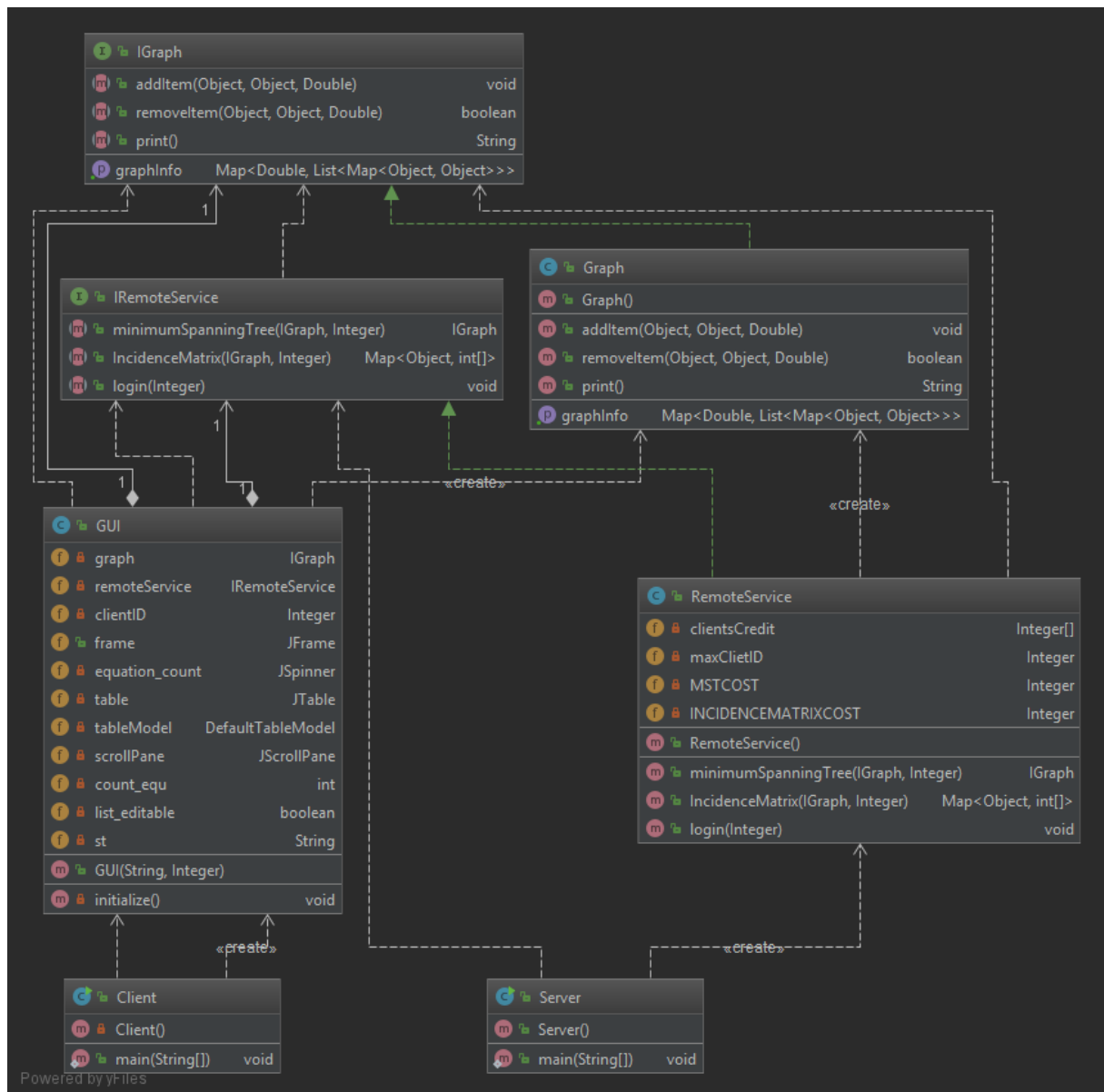
Client1 : 100 credit

Client2 : 200 credit

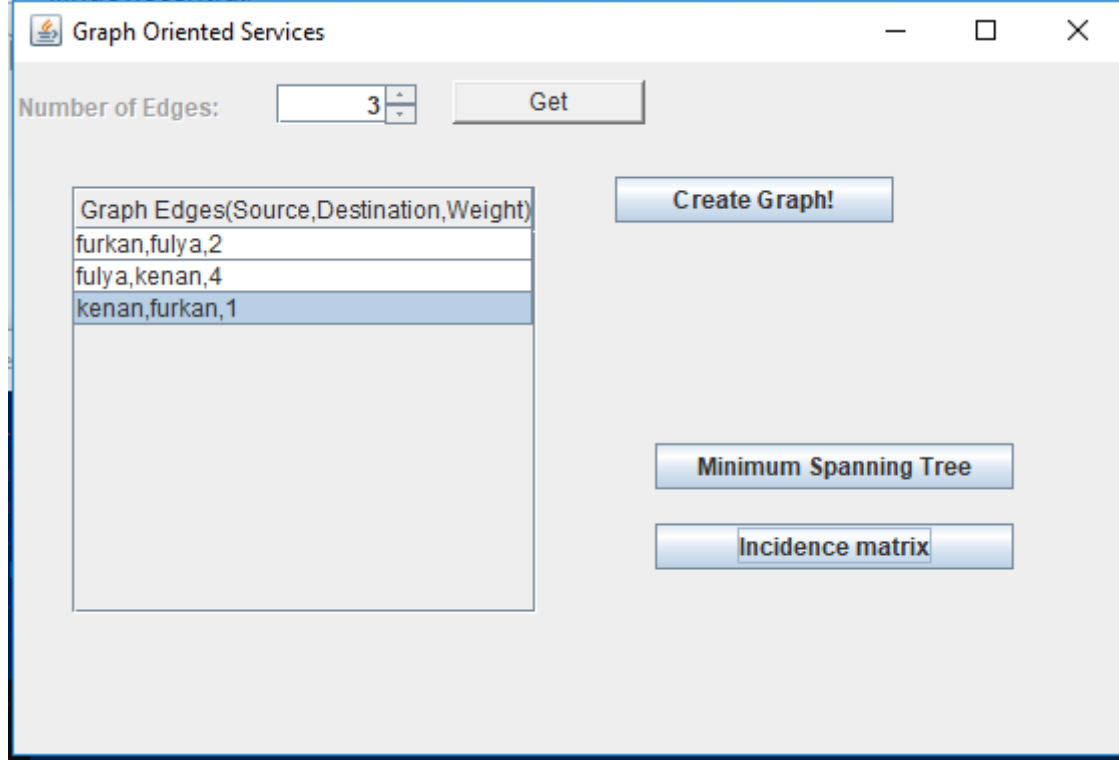
Client3 : 300 credit

Minimum Spanning Tree cost : 15 credit

Incidence Matrix cost : 10 credit



Kullanım:



- 1) Oluşturmak istenilen graph'ın edge sayısı girilir ve get butonuna basılır.
- 2) Açılan listede oluşturulmak istenen graph için her edgesi bir satıra yazılır. Vertexler arasında virgül koyulur ve yazım şekli şu şekildedir: "SourceVertex, DestinationVertex, Weight"
- 3) Create Graph! butonu ile graph oluşturulur.
- 4) Kullanılmak istenilen servisin(Minimum spanning Tree ya da Incidence matrix) butonuna basılır.

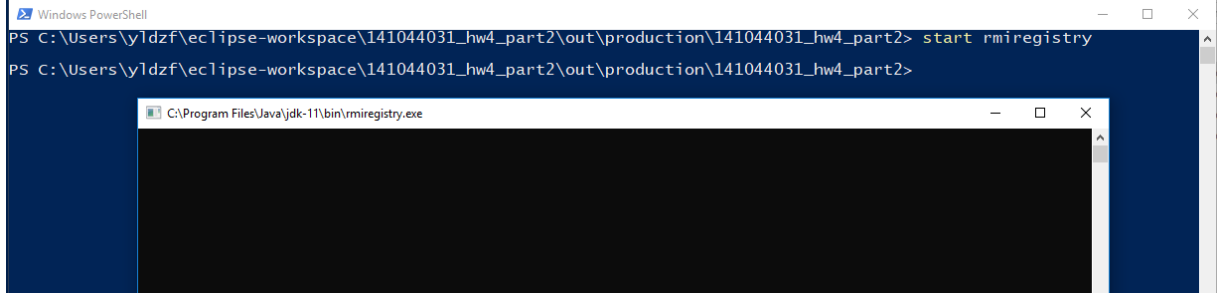
Not: GUI açılırken, servera bağlanmaya çalışır, eğer serverla bağlantı sağlanamazsa GUI açılmaz, hata penceresi açılır.

Çalıştırabilmek için

- 1) **Source file'ların class'larının olduğu path'de** yani "out\production\141044031_hw4_part2" directorysinde rmiregistry.exe başlatılır. Windows için bu komut "start rmiregistry" dir.
- 2) Server, Jarların olduğu dosyaya gelerek "java -jar Server.jar" komutu ile çalıştırılır.
- 3) Client jar dosyasına çift tıklayarak çalıştırılır.
 - a. Birden fazla client ile işlem yapılabilmesini gösterebilmek adına 3 client jar'ı oluşturuldu. Bu clientlerin birbirlerinden tek farkı cliedID leri. Server'da bu id lere göre hepsinin farklı credit'i mevcut.

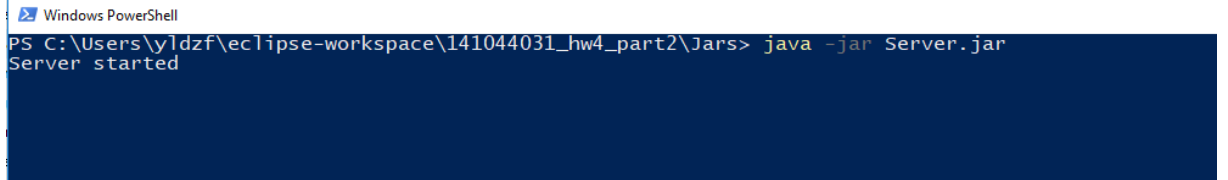
Server'a bağlantı olduğunda server'ın bunu "xxx id ye sahip client Server'a bağlandı" şeklinde bir bilgi gösterebilmek için remote servislerine "login" methodunu ekledim. Client başladığında GUI oluşturulurken, bu method'u client kendi ID sini vererek çağırıyor.

Class dosyalarının olduğu yerde rmiregistry başlatılır.



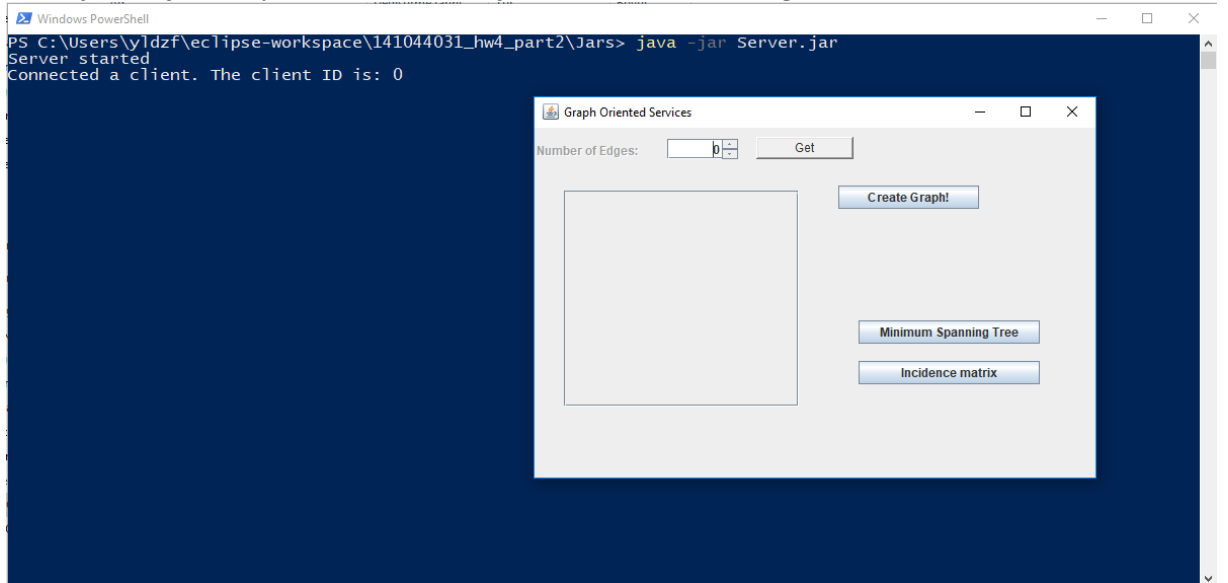
The screenshot shows a Windows PowerShell window with the command `start rmiregistry` executed. Below it, a small window titled `C:\Program Files\Java\jdk-11\bin\rmiregistry.exe` is open, displaying a black screen.

Jar dosyalarının bulunduğu klasörde “/Jars” server gösterildiği gibi başlatılır.



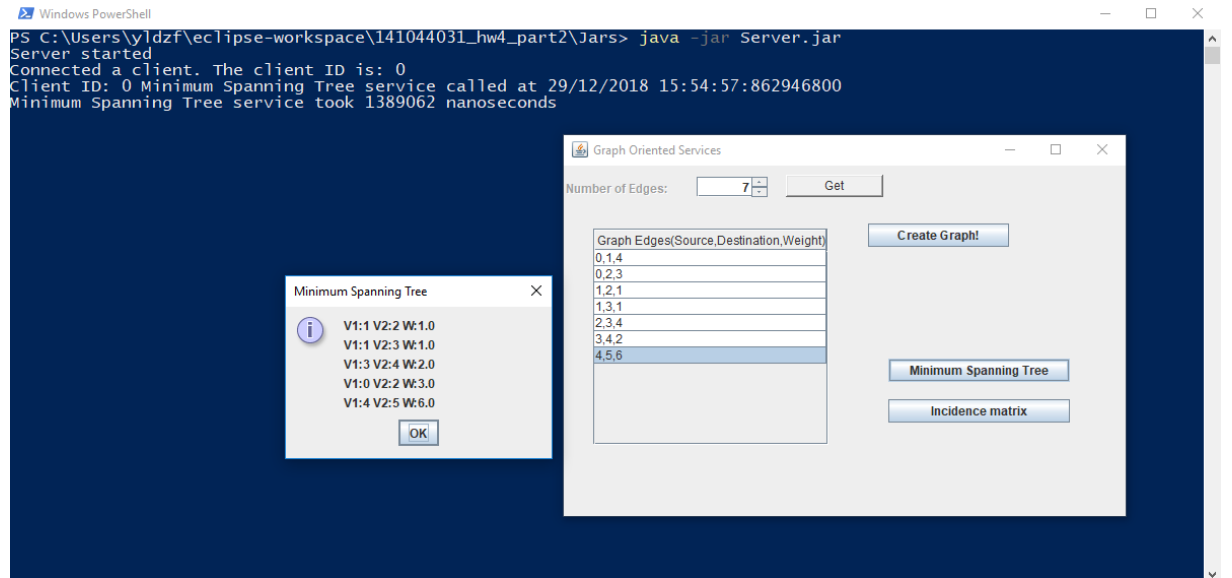
The screenshot shows a Windows PowerShell window with the command `java -jar Server.jar` executed. The output shows `Server started`.

Client jar’ına çift tıklayarak client GUI’si açıldı ve client servera bağlandı.

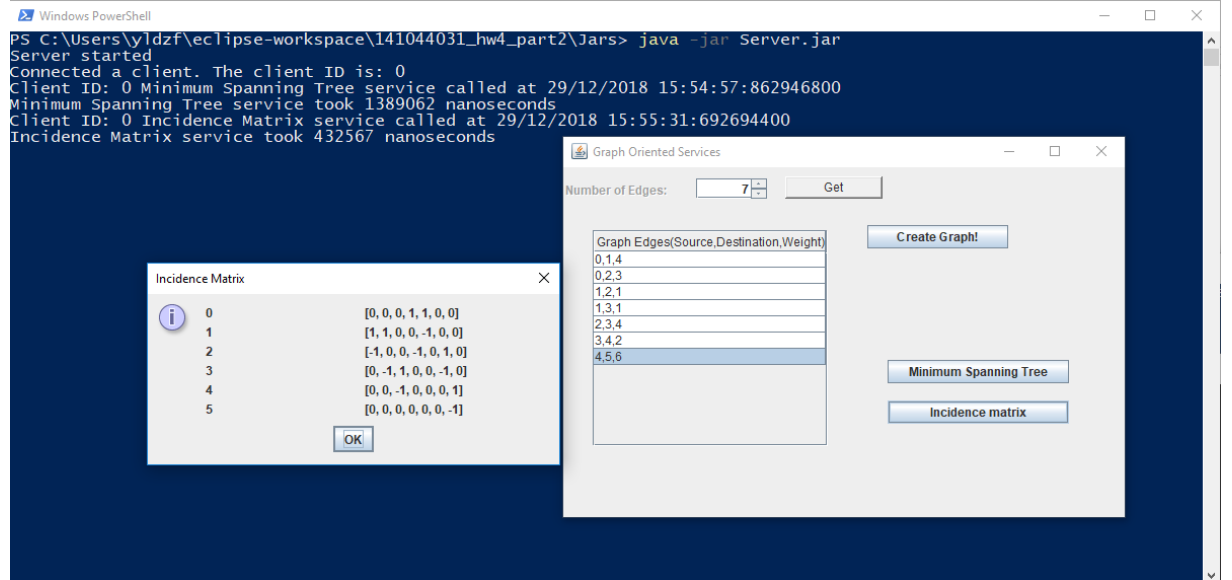


The screenshot shows a Windows PowerShell window with the command `java -jar Server.jar` executed. The output shows `Server started` and `Connected a client. The client ID is: 0`. Overlaid on the PowerShell window is a GUI titled `Graph Oriented Services`. The GUI has a text box for `Number of Edges:` with the value `0` and a `Get` button. Below this is a large empty box for the graph. To the right of the graph box are three buttons: `Create Graph!`, `Minimum Spanning Tree`, and `Incidence matrix`.

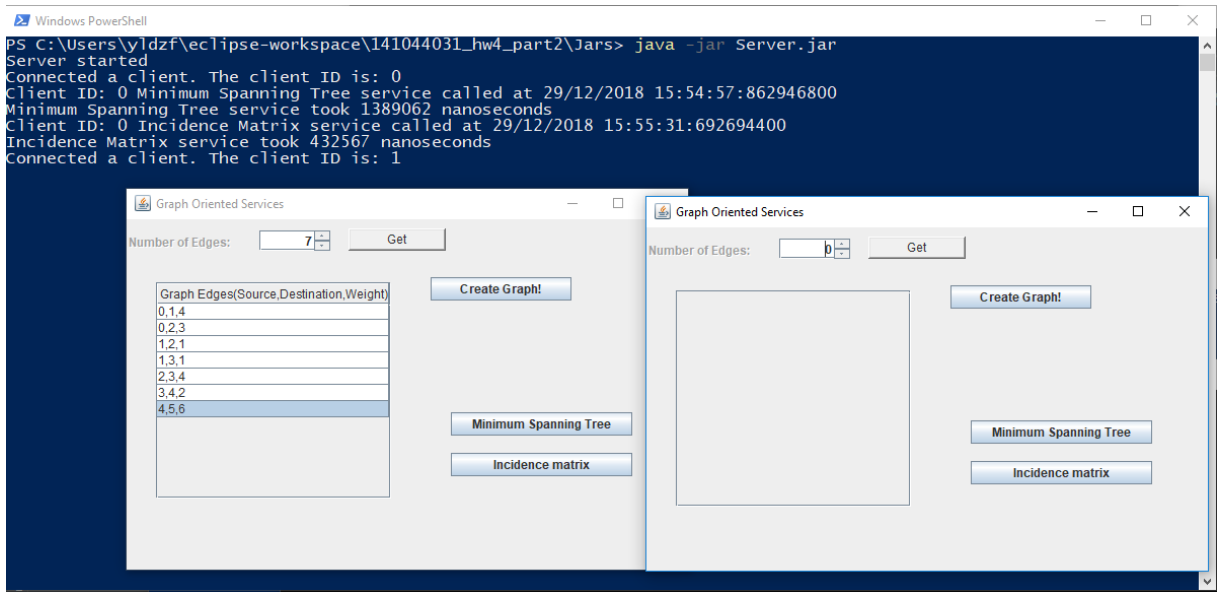
Graph oluşturularak, minimum spanning tree service çağırıldığında,



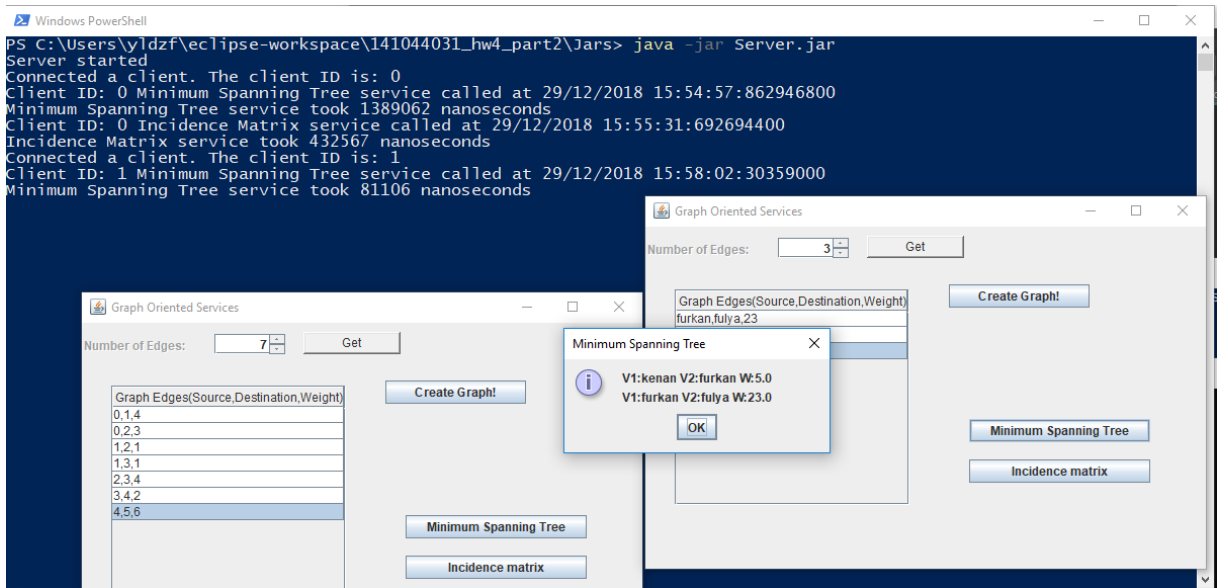
Incidence matrix service çağırıldığında,



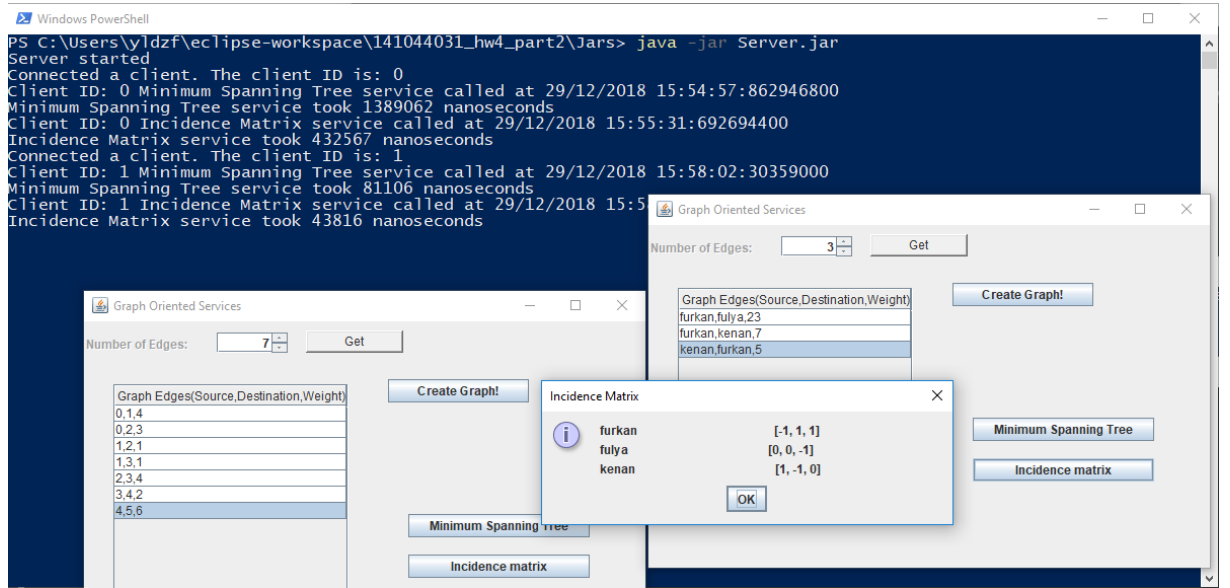
2. client açıldığında,



Client2'nin minimum spanning tree servicesi çağırıldığında,

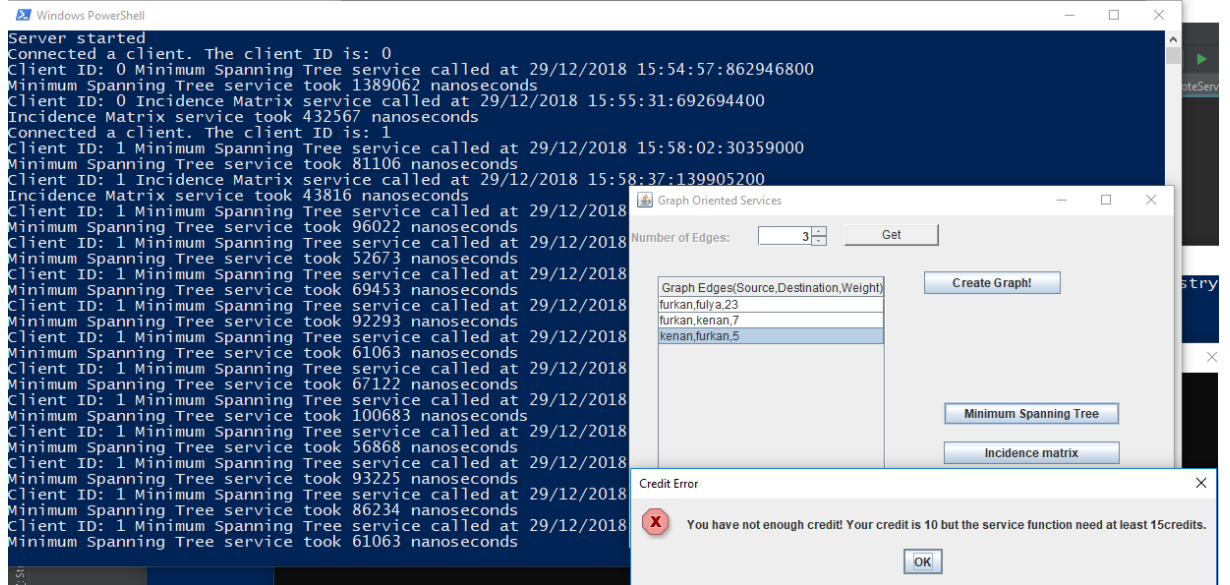


Client2'nin Incidence matrix'i çağırıldığında

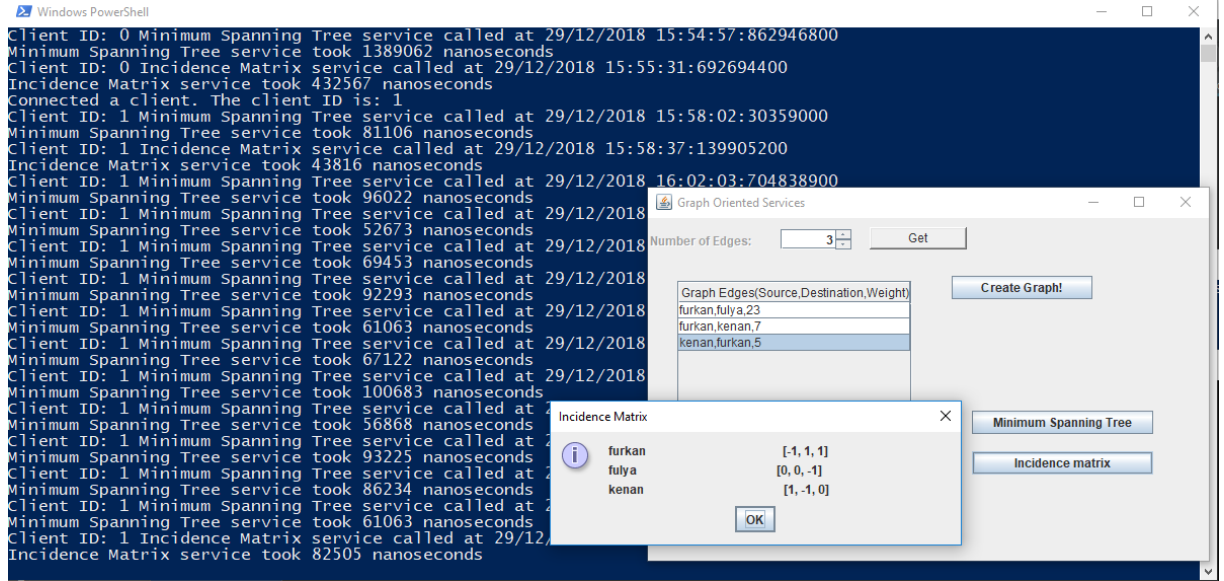


Matrix columnları sırasıyla şunları temsil ediyor,
Kenan'dan furkan'a bir edge var kenan:1, furkan:-1
Furkan'dan kenan'a bir edge var furkan:1, kenan:-1
Furkan'dan fulya'ya bir edge var furkan:1, fulya:-1

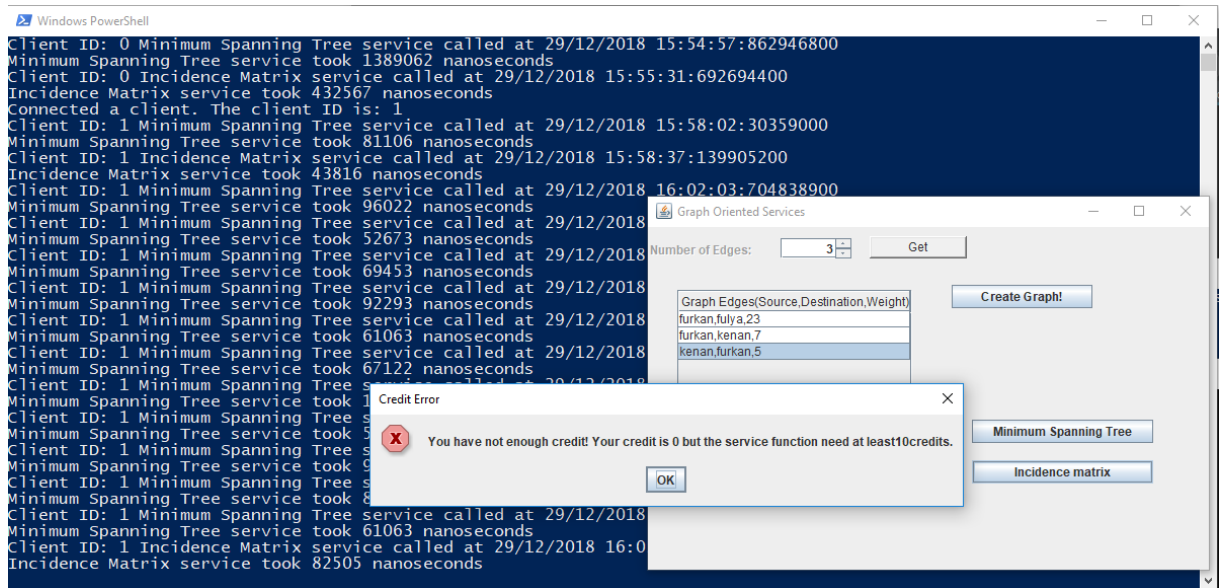
Client2 de, 12 kere minimum spanning tree, 1 kere incidence matrix çağırıldıktan sonra $12 \cdot 15 + 1 \cdot 10 = 190$ credit ediyor. Client2'nin 200 credit'i vardı geriye 10 credit kalıyor ve tekrar minimum spanning tree çağırıldığında bu 10 credit ile credit yetersiz uyarısını veriyor.



10 credit ile incidence matrix service'yi ise çalıştırabiliriz,



Ardından ise credit' imiz kalmadığından service çağıramayız,



Client1 ile işlem yapmaya devam edebiliriz,

The screenshot shows a Windows PowerShell window with the following output:

```
Incidence Matrix service took 43816 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:03:704838900
Minimum Spanning Tree service took 96022 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:05:846505100
Minimum Spanning Tree service took 52673 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:06:737369600
Minimum Spanning Tree service took 69453 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:07:628370100
Minimum Spanning Tree service took 92293 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:08:519180600
Minimum Spanning Tree service took 61063 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:09:409991100
Minimum Spanning Tree service took 67122 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:10:300801600
Minimum Spanning Tree service took 100683 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:11:191612100
Minimum Spanning Tree service took 56868 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:12:082422600
Minimum Spanning Tree service took 93225 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:12:973233100
Minimum Spanning Tree service took 86234 nanoseconds
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:13:864043600
Minimum Spanning Tree service took 61063 nanoseconds
Client ID: 1 Incidence Matrix service called at 29/12/2018 16:02:14:754854100
Incidence Matrix service took 82505 nanoseconds
Client ID: 0 Incidence Matrix service called at 29/12/2018 16:02:15:645664600
Incidence Matrix service took 130516 nanoseconds
Client ID: 0 Minimum Spanning Tree service called at 29/12/2018 16:02:16:536475100
Minimum Spanning Tree service took 198571 nanoseconds
Client ID: 0 Incidence Matrix service called at 29/12/2018 16:02:17:427285600
Incidence Matrix service took 283872 nanoseconds
Client ID: 0 Minimum Spanning Tree service called at 29/12/2018 16:02:18:318096100
Minimum Spanning Tree service took 326756 nanoseconds
```

The Graph Oriented Services application is also visible, showing the 'Number of Edges' set to 7. The 'Create Graph!' button is highlighted. The 'Minimum Spanning Tree' button is also visible. The 'Incidence matrix' button is also visible.

A 'Minimum Spanning Tree' dialog box is open, showing the following edges:

V1	V2	W
1	2	1.0
1	3	2.0
1	4	3.0
1	5	4.0
2	3	5.0

100 credit'i olan Client1 de 4 minimum spanning tre, 4 incidence matrix çağırıldığında $4*15 + 4*10 = 100$, service call yapmaya izin verilmeyecek. (Sağ alttaki Client1)

The screenshot shows a Windows PowerShell window with the following output:

```
Minimum Spanning Tree service called at 29/12/2018 16:02:19:209106600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:20:100017100
Minimum Spanning Tree service called at 29/12/2018 16:02:21:000027600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:21:900038100
Minimum Spanning Tree service called at 29/12/2018 16:02:22:800048600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:23:700059100
Minimum Spanning Tree service called at 29/12/2018 16:02:24:600069600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:25:500080100
Minimum Spanning Tree service called at 29/12/2018 16:02:26:400090600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:27:300101100
Minimum Spanning Tree service called at 29/12/2018 16:02:28:200111600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:29:100122100
Minimum Spanning Tree service called at 29/12/2018 16:02:30:000132600
Client ID: 1 Minimum Spanning Tree service called at 29/12/2018 16:02:30:900143100
Minimum Spanning Tree service called at 29/12/2018 16:02:31:800153600
Client ID: 1 Incidence Matrix service called at 29/12/2018 16:02:32:700164100
Incidence Matrix service took 258220 nanoseconds
Client ID: 0 Minimum Spanning Tree service called at 29/12/2018 16:02:33:600174600
Minimum Spanning Tree service took 198571 nanoseconds
Client ID: 0 Incidence Matrix service called at 29/12/2018 16:02:34:500185100
Incidence Matrix service took 283872 nanoseconds
Client ID: 0 Minimum Spanning Tree service called at 29/12/2018 16:02:35:400195600
Minimum Spanning Tree service took 326756 nanoseconds
Client ID: 0 Minimum Spanning Tree service called at 29/12/2018 16:02:36:300206100
Minimum Spanning Tree service took 302051 nanoseconds
Client ID: 0 Incidence Matrix service called at 29/12/2018 16:02:37:200216600
Incidence Matrix service took 84370 nanoseconds
```

The Graph Oriented Services application is also visible, showing the 'Number of Edges' set to 3. The 'Create Graph!' button is highlighted. The 'Minimum Spanning Tree' button is also visible. The 'Incidence matrix' button is also visible.

A 'Credit Error' dialog box is open, showing the following message:

You have not enough credit! Your credit is 0 but the service function need at least 15credits.

3. client ise seçildeki gibi bağlanarak işlem yapabiliyor

