Ahmet Mert Gülbahçe
141044015

# CSE443

HW4

Bu ödev için IDE kullanılmamıştır. Tüm testler **Ubuntu 18.04** işletim sisteminde, **openjdk version "10.0.2"** ile test edilmiştir. Test programlarını çalıştırmak için her bölümde bulunan **Çalıştırma Komutlarının** okunması önem arz etmektedir.

## PART 1

Bu bölümde, şema olarak verilen “Sonlu Durum Makinesi” için bir tasarım yapılmıştır. Tasarım ve kodun yazılması için **Durum** tasarım örüntüsü kullanılmıştır.

Her bir durumu ifade edecek **`State`** isminde bir arayüz oluşturulmuş ve mevcut tüm durumlar bu arayüzden türetilmiştir. Bu sınıflar aşağıdaki gibidir:

- `ChronicIllnessState`
- `FitState`
- `GraduateState`
- `NeedSleepState`
- `ReadyState`
- `UnableRodForAxeState`

Sonlu durum makinemizi temsil edecek olan **`StudentLifeCycle`** sınıfı, durumlar arasında verilen komuta göre hangi durumlar arasında geçiş yapabileceğimi belirler ve verilen şemanın uygulanabilmesini sağlar.

Durum sınıfları, verilen komuta cevap olarak komut geçerli ise ilk durum ve ikinci durum, geçerli değil ise verilen komutun geçersiz olduğunu ekrana yazdırır. Her durum ve komut için farklı cevap mesajları yazıldı fakat durumlar arası geçişler kolay anlaşılır olması için sadece durum geçişleri ve geçersiz komut mesajları kullanılmıştır.

### Çalıştırma Komutu

Terminal üzerinden part1 klasöründe “**make run**” komutu çalıştırılarak çalışma derlenir ve çalıştırılır.

Ahmet Mert Gülbahçe
141044015

# Çalışan Örnek

- Testin ilk bölümünde, Ready durumundan başlayıp gidilebilecek geçerli durumlar test edilmiştir.

```
--------------------
TESTING 'VALID' WAYS
--------------------
Default Start state: READY
--------------------
outTillLate
Ready --> Needing sleep
--------------------
sleep
Needing sleep --> Ready
--------------------
exercise
Ready --> Fit
--------------------
hardWork
Fit --> Graduate
--------------------
END
--------------------
```

- Diğer bölümlerde ise Ready, Fit ve Graduate durumları tüm komutlarla test edilmiştir. Bu işlemler için yedi farklı durum makinesi kullanılarak her durum için sonuçlar test edilmiştir.

```
--------------------
TESTING 'READY' STATE
--------------------
Default Start state: READY
--------------------
sleep
Can't sleep!
--------------------
exercise
Ready --> Fit
--------------------
hardWork
Ready --> Graduate
--------------------
outTillLate
Ready --> Needing sleep
--------------------
coffeeAndWork
Can't coffeeAndWork!
--------------------
buyingGTX1080
Ready --> Unable to become a rod for an axe
--------------------
cheating
Ready --> Unable to become a rod for an axe
--------------------
```

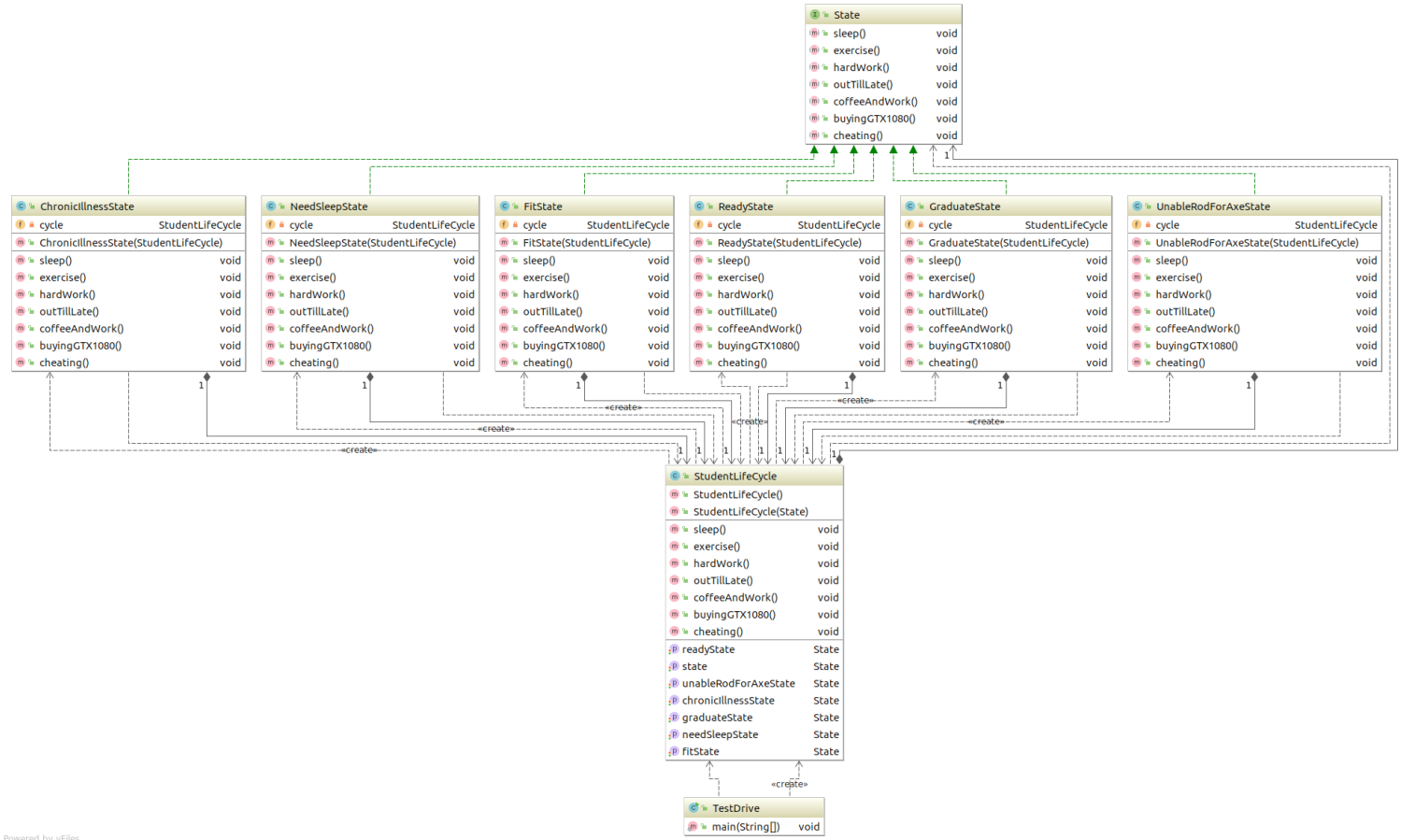
```
--------------------
Start state: FIT
--------------------
sleep
Can't sleep!
--------------------
exercise
Can't exercise!
--------------------
hardWork
Fit --> Graduate
--------------------
outTillLate
Can't outTillLate!
--------------------
coffeeAndWork
Can't coffeeAndWork!
--------------------
buyingGTX1080
Can't buyingGTX1080!
--------------------
cheating
Can't cheating!
--------------------
```

Ahmet Mert Gülbahçe
141044015

```
------------------------
TESTING 'GRADUATE' STATE
------------------------
Start state: GRADUATE
------------------------
sleep
Can't sleep!
------------------------
exercise
Can't exercise!
------------------------
hardWork
Can't hardWork!
------------------------
outTillLate
Can't outTillLate!
------------------------
coffeeAndWork
Can't coffeeAndWork!
------------------------
buyingGTX1080
Can't buyingGTX1080!
------------------------
cheating
Can't cheating!
------------------------
```

Ahmet Mert Gülbahçe
141044015

# UML Şeması



Ahmet Mert Gülbahçe
141044015

# PART 2

Bu bölüm, kendi içinde iki parçadan oluşmaktadır.

Yapılanlar:

- Directed ve weighted graph uygulaması (Node ve Edge ekleme ve terminal UI)
- RMI ile Server-Client modeli
- Server-Client üzerinde PRO sürüm, kullanıcı ve uygulama kredisi.

Yapılmayanlar:

- Graph algoritmaları
- Server üzerinde graph hesaplaması

# Graph

Bu bölümde, yönlü ve ağırlıklı graph yapısı uygulanmıştır. Graph içinde her bir node ile ilişkilendirilmiş bir komşuluk listesi bulunmaktadır. Graph içine eleman eklerken önce kurucu yöntem ile node sayısı verilir ve nodelar eklenir. Daha sonra kenarlar, Edge objesi olarak oluşturulur ve graph içine eklenir.

## Çalıştırma Komutu

Terminal üzerinden part2/graph klasörüne gelinmelidir. **“make”** komutu ile tüm dosyalar derlenmelidir.

Bu bölüm için iki farklı çalışma şekli bulunmaktadır.

1. İçinde beş adet şehrin ve aralarındaki uzaklığın bulunduğu önceden hazırlanmış test bulunmaktadır. Çalıştırmak için **“make test1”** komutu çalıştırılmalıdır.
2. Terminal üzerinde bulunan kullanıcı arayüzü ile graph oluşturmak için hazırlanmış bir test bulunmaktadır. Çalıştırmak için **“make test2”** komutu çalıştırılmalıdır.

Ahmet Mert Gülbahçe
141044015

# Çalışan Örnek

- Bu testte graph içine node ve edge bilgileri eklenmiştir. Test edilen şehirler ve aralarındaki mesafeler ile oluşturulmuş graph aşağıdaki gibidir.

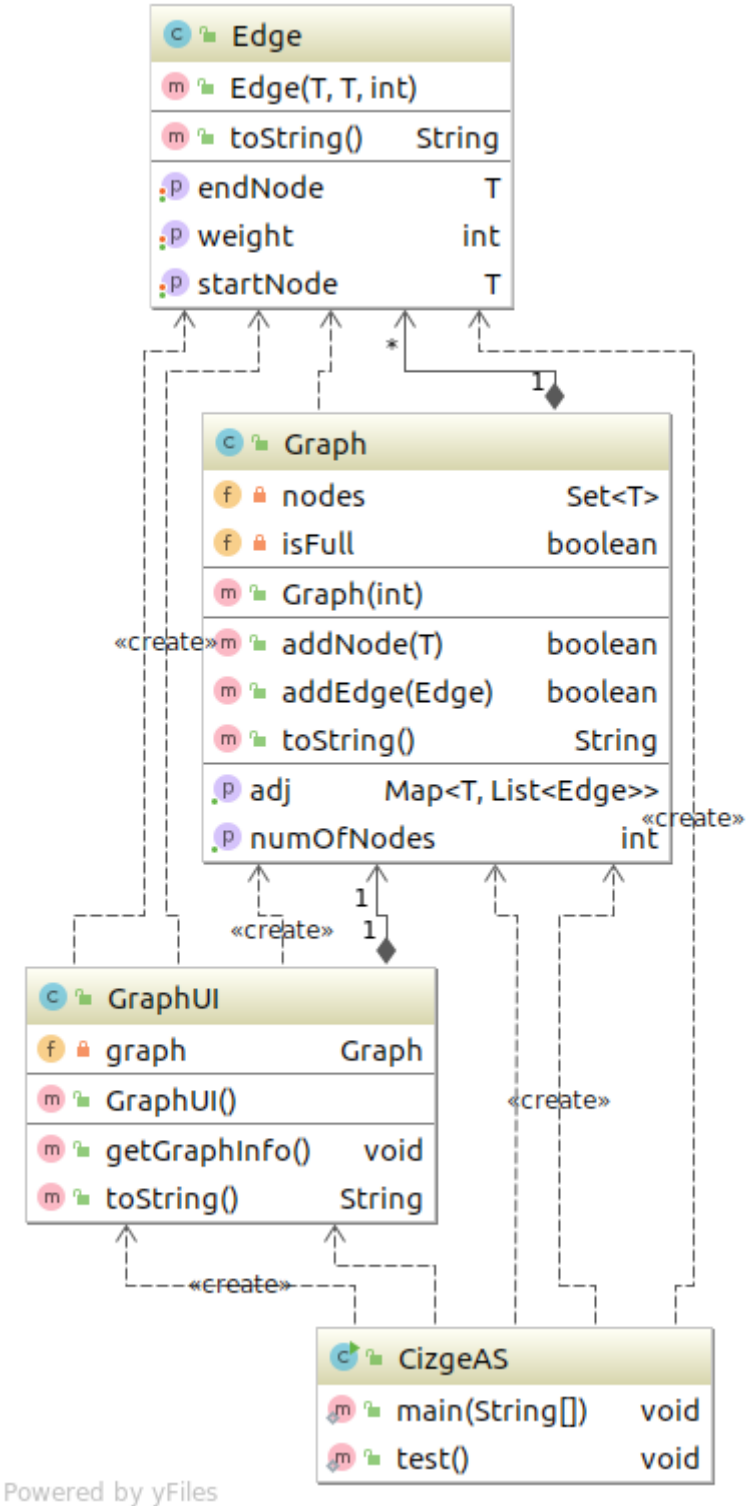
```
----------------
PREDEFINED TEST
----------------
Add Node:
----------------
-Ankara
-İstanbul
-Erzurum
-Adana
-Bursa
----------------
Add Edge:
----------------
-Erzurum Adana 806
-Erzurum İstanbul 1228
-İstanbul Ankara 453
-İstanbul Bursa 243
-Ankara Bursa 384
-Ankara Erzurum 875
-Adana Ankara 490
-Adana İstanbul 939
-Bursa Adana 839
-Bursa Erzurum 1239
----------------
----------------
----------------
Node: Adana
        -Adana --> Ankara : 490
        -Adana --> İstanbul : 939
Node: İstanbul
        -İstanbul --> Ankara : 453
        -İstanbul --> Bursa : 243
Node: Bursa
        -Bursa --> Adana : 839
        -Bursa --> Erzurum : 1239
Node: Erzurum
        -Erzurum --> Adana : 806
        -Erzurum --> İstanbul : 1228
Node: Ankara
        -Ankara --> Bursa : 384
        -Ankara --> Erzurum : 875
```

- İkinci test durumunda ise program başladıktan sonra terminal ekranındaki UI ile graph örneğinizi oluşturabilirsiniz.

```
----------------
TEST YOURSELF
----------------
Number of nodes: 2
Add node: İstanbul
Add node: Ankara
Number of edges: 2
Add edge:
-Enter start node: İstanbul
-Enter end node: Ankara
-Enter weight between nodes: 453
Add edge:
-Enter start node: Ankara
-Enter end node: İstanbul
-Enter weight between nodes: 453
Node: İstanbul
        -İstanbul --> Ankara : 453
Node: Ankara
        -Ankara --> İstanbul : 453
```

Ahmet Mert Gülbahçe
141044015

# UML Diagram



Ahmet Mert Gülbahçe
141044015

# Remote

Bu bölümde, Java'nın sağladığı RMI kütüphaneleri kullanılarak server-client modelinde çalışan bir hesap makinesi arası geliştirilmiştir.

Geliştirilen hesap makinesine, kullanıcıların kendi cihazında çalıştırdığı program üzerinden giriş yapması veya kayıt olması gerekmektedir. Kayıt olan her bir kullanıcıya, sistem tarafından 10 hesaplama kredisi tanımlanır ve her bir işlem yaptıkça bu kredilerinden kullanılır.

## Çalıştırma Komutu

Terminal üzerinden part2/remote klasörüne gelinmelidir. **"make"** komutu ile tüm dosyalar derlenmelidir. Ardından en az üç terminal penceresi açılmalıdır.

1. Birinci pencerede **"make reg"** komutu çalıştırılır.
2. İkinci pencerede **"make server"** komutu çalıştırılır.
3. Diğer pencerelerde **"make client"** komutu çalıştırılarak server ve client çalışır hale getirilir.

## Çalışan Örnek

- RMI Registry çalışır halde beklemektedir.

```
mert@mert-P7U:~/Desktop/part2/remote$ make
javac *.java
rmic Server
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
mert@mert-P7U:~/Desktop/part2/remote$ make reg
rmiregistry
```

- Server çalıştırılır ve server tarafından hazır olduğu bilgisi alınır.

```
mert@mert-P7U:~/Desktop/part2/remote$ make server
java Server
SERVER READY!
```

- Client çalıştırılır ve server ile bağlantısı sağlanır. Terminal üzerindeki kullanıcı arayüzü ile kayıt ve giriş işlemlerini yapar. Yeni kayıt olduğu için sistem tarafından 10 kredi verilir.

```
mert@mert-P7U:~/Desktop/part2/remote$ make client
java Client
##############################
#        Calculator          #
##############################
# 1-login                    #
# 2-Register                 #
##############################
2
REGISTER
Username:
ahmet
Password:
123
Success!
Enter first operand: 12
Enter second operand: 12
Select operation(+,-,*): *
12*12=144

You have 9 credits!
```

- Kullanıcı giriş yapmıştır ve kredisi olmadığından dolayı sistem tarafından uyarı alır ve hesaplama işlemi yapamaz.

```
mert@mert-P7U:~/Desktop/part2/remote$ make client
java Client
##############################
#        Calculator          #
##############################
# 1-login                    #
# 2-Register                 #
##############################
1
LOGIN
Username:
ahmet
Password:
123
Success!

You have 0 credits!

Sorry, you don't have enough credit to use calculator!
You have 0 credits!
```

Ahmet Mert Gülbahçe
141044015

# UML Diagram