

# T.C. SAKARYA ÜNİVERSİTESİ

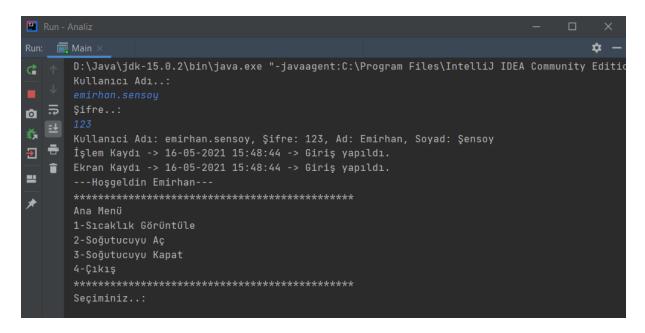
# BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESI BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

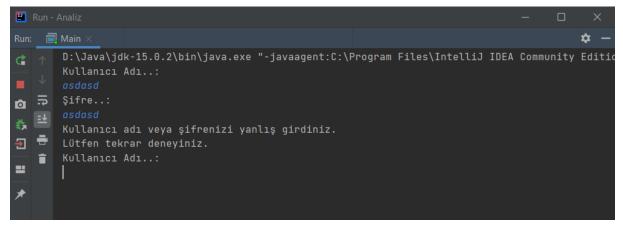
# NESNE YÖNELİMLİ ANALİZ VE TASARIM PROJE RAPORU

Emirhan ŞENSOY
G191210040
2.Öğretim B Grubu
emirhan.sensoy@ogr.sakarya.edu.tr

## Kullanıcı Doğrulama Ekranı

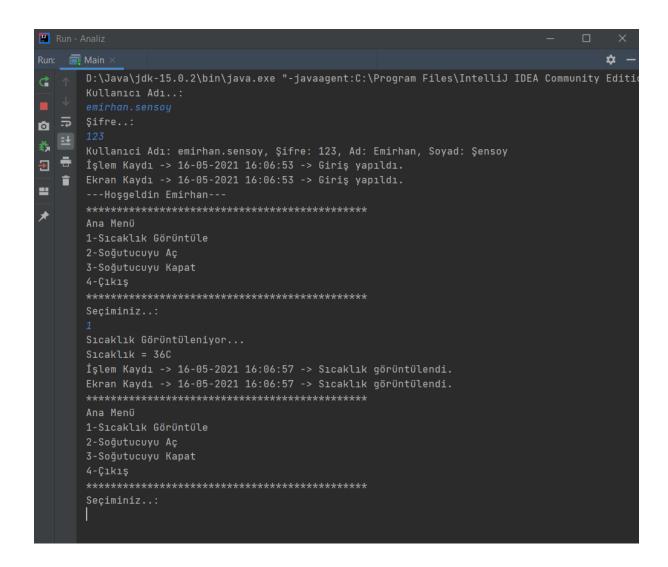
Kullanıcı doğrulama ekranı, girilen kullanıcı adı ve şifreyi alıp akıllı cihaz bilgi sisteminden postgreSQL sunucuya erişerek veritabanında verilerin doğru girilip girilmediğini kontrol eder ve giriş işlemini gerçekleştirir.





#### Sıcaklık Görüntüleme Ekranı

Sıcaklık görüntüleme ekranı, ağ arayüzü aracılığı ile algılayıcıdan aldığı sıcaklık değerini ekrana yazdırır.



### Soğutucu Açma Ekranı

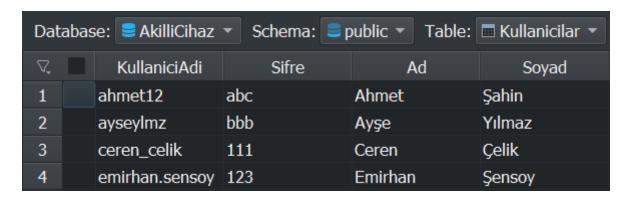
Soğutucu açma ekranı, ağ arayüzü aracılığı ile eyleyici üzerinden soğutucu durumunu kontrol edip soğutucu açma işlemini gerçekleştirir.

## Soğutucu Kapatma Ekranı

Soğutucu kapatma ekranı, ağ arayüzü aracılığı ile eyleyici üzerinden soğutucu durumunu kontrol edip soğutucu kapatma işlemini gerçekleştirir.

```
D:\Java\jdk-15.0.2\bin\java.exe "-javaagent:C:\Program Files\IntelliJ IDEA Community Editi
       Kullanıcı Adı..:
Ō
   E±
  च İşlem Kaydı -> 16-05-2021 16:14:41 -> Giriş yapıldı.
==
       Ana Menü
       1-Sıcaklık Görüntüle
       2-Soğutucuyu Aç
       3-Soğutucuyu Kapat
       4-Çıkış
       ****************
       İşlem Kaydı -> 16-05-2021 16:14:45 -> Soğutucu kapat çalıştırıldı.
       Ana Menü
       1-Sıcaklık Görüntüle
       2-Soğutucuyu Aç
```

#### Veritabanı Görüntüsü



# **Dependency Inversion İlkesi**

Dependency inversion ilkesi bağımlılığın ters çevrilmesi prensibi olarak da bilinir. Mantık olarak üst ve alt sınıfların birbirlerine bağımlı olmalarına karşı çıkan bir prensiptir. Bu bağımlılık arayüzler ile giderilir. Üst sınıfın, alt sınıfın implement ettiği arayüzü kullanması sağlanarak iki sınıf arasındaki bağımlılık azaltılır. Amaç projelerdeki zaman maliyetini düşürmek ve projeye sonradan eklenecek yapıların projeye girişini, okunabilirliğini ve yenilenebilirliğini artırmaktır.

```
package mypackage;

public class SicaklikGoruntuleme implements IIslem

{
    IEkran ekran;
    IAlgilayici algilayici;
    IAgArayuzu agArayuzu;

private SicaklikGoruntuleme(SicaklikGoruntulemeBuilder builder)

{
    this.ekran = builder.ekran;
    this.algilayici = builder.algilayici;
    this.algilayici = builder.agArayuzu;

}

@Override
public void islemYap()

{
    Araclar.bekle( sure: 600);
    ekran.mesajGoruntule(agArayuzu.sicaklikGonder(algilayici.sicaklikOku()));
    Araclar.bekle( sure: 500);
}
```

```
Private void islemSmcimi()

{

int secim;

do

4

secim = anaMenuyuGoster();

switch (secim)

{

case SIGAKLIK_GORUNITULENE -> {

IIslem sicakLikGoruntuleme = new SicakLikGoruntulemeSuilder(ekran, algilayici, agArayuzu).build();

sicakLikGoruntuleme.islemTap();

publisher.notify(m "SicakLik goruntulemi.");

}

case SOGUTUCU_AC -> {

IIslem sogutucuAc = new SogutucuAc.SogutucuAcBuilder(ekran, eyleyici, agArayuzu).build();

sogutucuAc.islemTap();

publisher.notify(m "Sigutucu ac calistiridi.");

}

case SOGUTUCU_KAPAT -> {

IIslem sogutucuAc = new SogutucuKapat.SogutucuKapatBuilder(ekran, eyleyici, agArayuzu).build();

sogutucuRapat.islemTap();

publisher.notify(m "Sigutucu kapat calistiridi.");

}

case CINIS -> {

ekran.mesajGoruntule("Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");

publisher.notify(m "Cikis yapilyor...");
```

#### **Builder Deseni**

Builder tasarım deseni, uzun ve kompleks kodu kısaltıp kolaylaştırmaya yarayan bir tasarım desenidir. Geliştirilen yazılımda farklı parametre kombinasyonlarına sahip birden çok kurucu fonksiyon veya çok fazla parametre alan bir obje varsa kullanılır.

#### Observer Deseni

Observer adı üzerinde gözlemci, izleyici, gözcü yahut gözetmen diye nitelendirilen, anlamı gibi işlev gören bir tasarım desenidir. Mevcut bir nesnenin durumunda herhangi bir değişiklik olduğunda, bu değişiklerden diğer nesneleri haberdar etmeye yarar. Observer tasarım deseninde; takip edilecek nesneye subject, subject'i takip eden nesnelere concrete ve concrete nesnelerin birden fazla tip olmasını sağlayan arayüze de observer denir.

```
Isubjectjava ×

package mypackage;

public interface Isubject

void attach(IObserver o);
void detach(IObserver o);
void notify(String m);

}
```

```
package mypackage;

public interface IObserver

void update(String m);

}
```

```
package mypackage;

import java.text.SimpleDateFormat;
import java.util.Date;

public class EkranKaydi implements IObserver

{
    Date date;
    SimpleDateFormat formatter;

    public EkranKaydi()
    {
        formatter = new SimpleDateFormat( pattern: "dd-MM-yyyy HH:mm:ss");
}

@
@Override
public void update(String mesaj)

date = new Date();
    System.out.println("Ekran Kayd1 -> " + formatter.format(date) + " -> " + mesaj);
}

and
y

date = new Date();
    System.out.println("Ekran Kayd1 -> " + formatter.format(date) + " -> " + mesaj);
}
```

```
package mypackage;

import java.text.SimpleDateFormat;
import java.util.Date;

public class IslemKaydi implements IObserver
{
    Date date;
    SimpleDateFormat formatter;

public IslemKaydi()
{
    formatter = new SimpleDateFormat( pattern: "dd-MM-yyyy HH:mm:ss");
}

@Override
public void update(String mesaj)
{
    date = new Date();
    System.out.println("işlem Kaydı -> " + formatter.format(date) + " -> " + mesaj);
}
}
```

```
package mypackage;

public class Uygulama

public static void main(String[] args)

Publisher publisher = new Publisher();

IslemKaydi islemKaydi = new IslemKaydi();

EkranKaydi ekranKaydi = new EkranKaydi();

publisher.attach(islemKaydi);

publisher.attach(ekranKaydi);

AkilliCihaz cihaz = new AkilliCihaz(publisher);

cihaz.calistir();

}
```

# **Proje Github Linki**

 $\frac{https://github.com/emirhansensoy/ObjectOrientedAnalysisAndDesign-\\SmartDeviceProject}{}$ 

# Projenin Anlatıldığı Youtube Video

https://youtu.be/8ILPbymZOEg