Smart Device System
(Akıllı Cihaz Yönetim Sistemi)

Grup Üyeleri:

Özgür Taşkıran (No: 23181616016)
Murat Efe Yalnız (No: 23181617006)

Ders Adı:

Nesne Yönelimli Programlama

Dönem Bilgisi:

2024 Bahar Dönemi

İçindekiler

1. Giriş

1.1 Projenin Amacı

SmartDeviceSystem, kullanıcıların sahip oldukları farklı türdeki akıllı ev cihazlarını tek bir platform üzerinden kolayca yönetmelerini ve kontrol etmelerini sağlamaya yönelik bir sistemdir. Bu proje, hem uzaktan erişim imkânı sağlar hem de ev içerisinde verimli bir cihaz yönetimi sunar. Kullanıcılar, cihazların özelliklerini özel ayarlamalarla optimize edebilir ve daha konforlu bir yaşam alanı oluşturabilir.

1.2 Projenin Kapsamı

- Kullanıcı giriş sistemi (kaydolma ve giriş yapma).
- Farklı türdeki akıllı cihazları ekleme ve yönetme.
- Cihazlara özel kontrol mekanizmalarının sağlanması (parlaklık ayarı, sıcaklık kontrolü vb.).
- Kullanıcı dostu bir grafiksel arayüz ile etkileşim.

1.3 Kullanılan Teknolojiler

- Programlama Dili: Java (JDK 24)
- Arayüz Çerçevesi: Java Swing
- Proje Yönetimi: GitHub
- IDE ve Geliştirme Ortamı: IntelliJ IDEA Ultimate Edition

2. Teknik Bölüm

2.1 Problemin Tanımı

Modern evlerde, farklı markaların çeşitli akıllı cihazları bulunmaktadır. Ancak bu cihazları kontrol etmek için farklı uygulamaların kullanılması gerekebilir. Bu durum, hem zorluk yaratmakta hem de zaman kaybına neden olmaktadır. SmartDeviceSystem projesi, bu sorunu çözmek amacıyla tüm cihazları tek bir platformdan yönetebilme imkânı sunar.

2.2 Kullanılan Sınıflar, Metotlar ve Mimari

Bu proje, "SmartDevice" adlı temel sınıf üzerinden bir yapı inşa edilmiştir. Her bir cihaz, SmartDevice sınıfını miras alarak kendine özgü fonksiyonları ve özellikleriyle entegre çalışır. Bu yapı, cihazlar arasındaki etkileşimi ve kontrolü düzenlerken kullanıcıya kolay bir yönetim ve izleme deneyimi sunmayı hedefler.

DoorLock

- Metotlar:
  - setPassword(String password): Kapı kilidi için şifre belirler.
  - unlock(String password): Doğru şifreyle kapıyı kilitsiz hale getirir.
  - lock(): Kapıyı kilitler.
  - isLocked(): Kapının kilitli olup olmadığını kontrol eder.

Fridge

- Metotlar:
  - setUpperTemperature(double temperature): Üst bölme sıcaklığını ayarlar.
  - setLowerTemperature(double temperature): Alt bölme sıcaklığını ayarlar.
  - getUpperTemperature(), getLowerTemperature(): Sıcaklık değerlerini döner.
  - getAverageTemperature(): Ortalama sıcaklığı hesaplar.

Light

- Metotlar:
  - getBrightness(), setBrightness(int brightness): Parlaklık seviyesini okur ve ayarlar.
  - setRGBColor(int red, int green, int blue): Işığın rengini belirler.
  - turnOn(), turnOff(): Işığı açar veya kapatır.

RobotVacuum

- Metotlar:
  - startCleaning(), stopCleaning(): Temizlik işlemine başlar ve bitirir.
  - listRooms(): Temizlik için mevcut odaları listeler.
  - selectRoom(String roomCode): Temizlik için bir oda seçer.
  - isVacuumRunning(): Temizlik durumunu kontrol eder.
  - getSelectedRoom(): Seçili odanın bilgisini döner.

SmartCurtain

- Metotlar:
  - turnOn(), turnOff(): Perdeyi açar veya kapatır.
  - setOpenPercentage(int percentage): Perdenin açılma oranını ayarlar.
  - getOpenPercentage(): Perdenin mevcut açıklık oranını döner.

Television

- Metotlar:
  - isOn(), turnOn(), turnOff(): Televizyonun durumunu kontrol eder ve açıp kapatır.
  - getVolumeLevel(), setVolume(int volume): Ses seviyesini okur ve ayarlar.
  - getCurrentChannel(), changeChannel(int channelIndex): Mevcut kanalı görüntüler ve kanal değiştirir.
  - getChannelList(): Kanal listesini döner.

Thermostat

- Metotlar:
  - turnOn(), turnOff(): Termostatı açar veya kapatır.
  - setTemperature(double temperature): Sıcaklık ayarı yapar.
  - getCurrentTemperature(): Mevcut sıcaklığı döner.

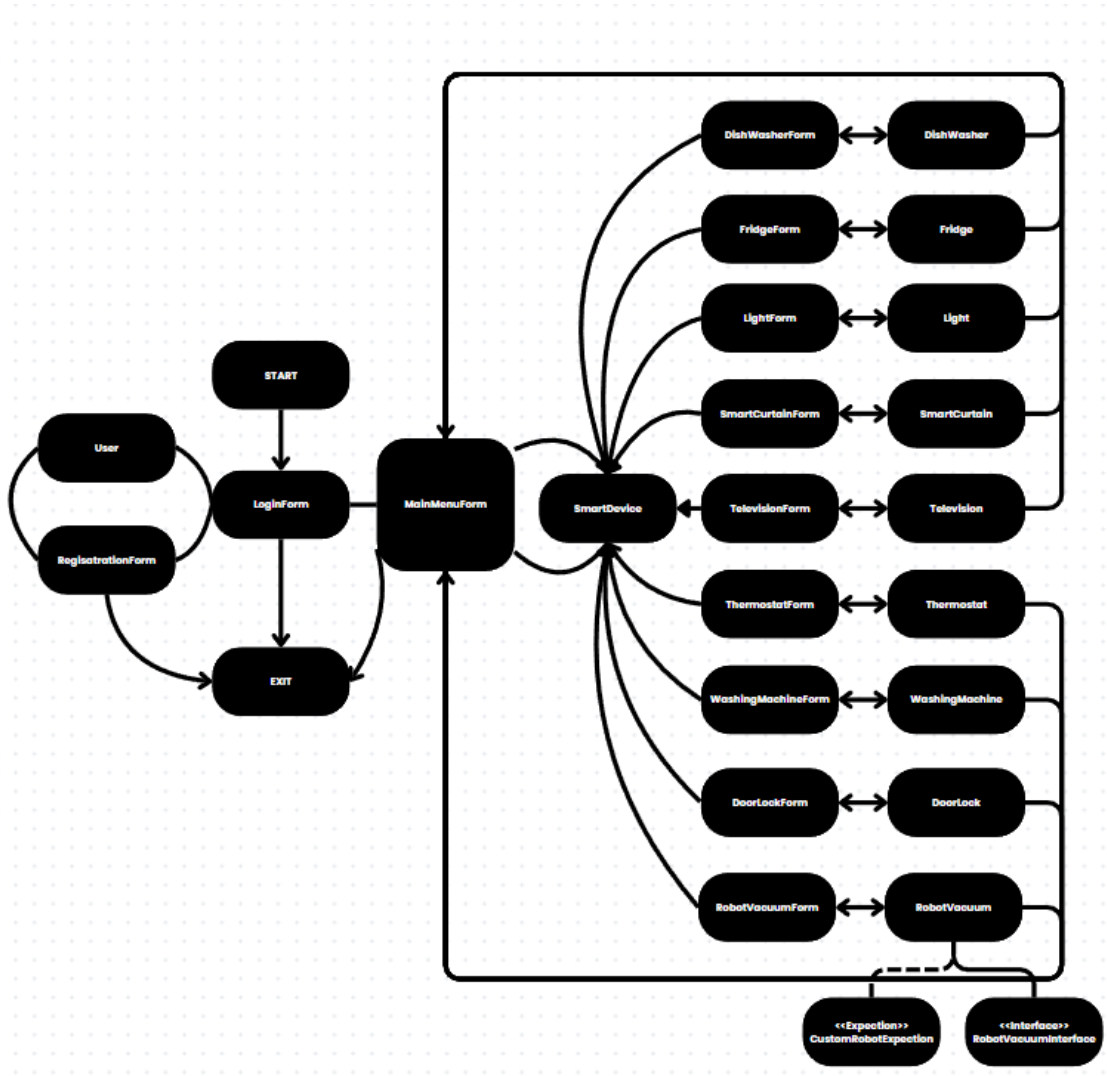
WashingMachine

- Metotlar:
  - isRunning(): Çamaşır makinesinin çalışıp çalışmadığını kontrol eder.
  - getSelectedProgram(), setProgram(String programCode): Seçili programı görüntüler ve yeni bir program ayarlar.
  - start(), stop(), completeProgram(): Programı başlatır, durdurur veya tamamlar.

DishWasher

- Metotlar:
  - startCleaning(), stopCleaning(): Temizlik işlemine başlar ve sonlandırır.
  - listRooms(), selectRoom(): Temizlik için odaları listeler ve seçer.
  - isVacuumRunning(): İşlem durumunu kontrol eder.

Projenin mimarisi, her cihazın bağımsız ama uyumlu bir şekilde çalışmasını sağlayacak şekilde tasarlanmıştır. Aşağıda verilmiş olan şekil 1'deki Java UML diyagramında bu yapının görsel temsilini bulabilirsiniz.



Şekil 1: Smart Device System UML Diagramı

2.3 Kod Parçaları ve Açıklamalar

SmartDevice sınıfı, akıllı cihazların temel işlevlerini yönetmek için tasarlanmış soyut bir sınıftır. Bu sınıf, cihazın adı ve açık/kapalı durumu gibi temel özellikleri tutar. Cihaz adı yalnızca okunabilir (readonly) olarak tanımlanmış, yani sadece sınıf içinde atanabilir ve dışarıdan değiştirilemez. Cihazın açık veya kapalı durumu ise boolean bir değerle tutulur ve sınıf içinde değiştirilebilir. SmartDevice sınıfı, cihazı açma ve kapama işlevlerini sunar. Ayrıca, cihazın durumu (açık/kapalı) dışarıdan erişilebilir, ancak yalnızca sınıf içinde değiştirilmesi sağlanmıştır. Bu sınıf, tüm akıllı cihazlar için temel işlevselliği sağlayan bir şablon işlevi görür ve bu sınıfı miras alan diğer cihaz sınıfları (örneğin, televizyon, çamaşır makinesi) daha spesifik işlevlerle bu temel işlevleri genişletebilir.

```
package Devices;

public abstract class SmartDevice {

    private final String name; // Cihazın adı (readonly)
    private boolean status; // Cihazın açık/kapalı durumu

    public SmartDevice(String name, boolean status) {
        this.name = name;
        this.status = status;
    }

    // Cihaz adı (readonly getter)
    public String getName() {
        return name;
    }

    // Cihaz durumu (getter)
    public boolean getStatus() {
        return status;
    }

    // Cihaz durumu (setter private yapılmış: sadece sınıf içinde
değişebilir)
    protected void setStatus(boolean status) {
        this.status = status;
    }

    // Cihazı aç
    public void turnOn() {
        this.status = true;
    }

    // Cihazı kapat
    public void turnOff() {
        this.status = false;
    }
}
```

Light sınıfı, ışıkların parlaklık seviyesi ve RGB renk ayarlarını yönetmek için tasarlanmıştır. Bu sınıf, SmartDevice sınıfından türetilmiş olup temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, parlaklık seviyesini ayarlayabilir, RGB renklerini değiştirebilir ve cihazı açıp kapatabilir. Kod, hata kontrol mekanizmalarıyla donatılmıştır ve cihazın durumunu yönetmek için çeşitli metotlar sunar.

```
package Devices;
public class Light extends SmartDevice {

    private int brightness; // Işığın parlaklık seviyesi (0-100 arası
bir değer)
    private int[] rgbColor = {255, 255, 255}; // RGB rengi (varsayılan
olarak beyaz [255, 255, 255])
    public Light(String name, boolean status) {
        super(name, status);
        this.brightness = 100; // Varsayılan parlaklık
    }
    // Parlaklık seviyesi (getter / setter)
    public int getBrightness() {
        return brightness;
    }
    public void setBrightness(int brightness) {
        if (brightness < 0 || brightness > 100) {
            throw new IllegalArgumentException("Parlaklık seviyesi 0 ile
100 arasında olmalıdır.");
        }
        this.brightness = brightness;
    }
    // RGB renk ayarları (getter / setter)
    public void setRGBColor(int red, int green, int blue) {
        if (red < 0 || red > 255 || green < 0 || green > 255 || blue < 0
|| blue > 255) {
            throw new IllegalArgumentException("RGB değerleri 0 ile 255
arasında olmalıdır.");
        }
        this.rgbColor[0] = red;
        this.rgbColor[1] = green;
        this.rgbColor[2] = blue;
    }
    // Cihazı aç
    @Override
    public void turnOn() {
        super.turnOn(); // SmartDevice'deki temel işlemleri çağır
        this.setStatus(true); // Işık durumunu aç
    }
    // Cihazı kapat
    @Override
    public void turnOff() {
        super.turnOff(); // SmartDevice'deki temel işlemleri çağır
        this.setStatus(false); // Işık durumunu kapat
    }
}
```

Fridge sınıfı, akıllı buzdolabı cihazlarının üst ve alt bölme sıcaklıklarını ayrı ayrı yönetmek için tasarlanmıştır. Bu sınıf, SmartDevice sınıfından türetilmiş olup temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, üst ve alt bölmenin sıcaklıklarını ayarlayabilir, belirli sınırlar içinde kontrol edebilir ve ortalama sıcaklık değerini hesaplayabilir. Kod, geçersiz sıcaklık değerlerini kontrol eden hata yönetimi mekanizmalarıyla donatılmıştır ve cihazın verimli bir şekilde yönetilmesini sağlayan çeşitli metotlar sunar.

```
package Devices;
public class Fridge extends SmartDevice {
    private double upperCompartmentTemperature;   // Üst bölmenin
sıcaklığı
    private double lowerCompartmentTemperature;   // Alt bölmenin
sıcaklığı
    private static final double MIN_TEMPERATURE = -20.0; // Minimum
sıcaklık (-20 °C)
    private static final double MAX_TEMPERATURE = 10.0;  // Maksimum
sıcaklık (+10 °C)
    // Constructor
    public Fridge(String name, boolean status) {
        super(name, status);
        this.upperCompartmentTemperature = 0.0; // Varsayılan başlangıç
sıcaklığı
        this.lowerCompartmentTemperature = 0.0; // Varsayılan başlangıç
sıcaklığı}
    // Üst bölmenin sıcaklığı
    public double getUpperCompartmentTemperature() {
        return upperCompartmentTemperature;}
    public void setUpperCompartmentTemperature(double value) {
        if (value >= MIN_TEMPERATURE && value <= MAX_TEMPERATURE) {
            this.upperCompartmentTemperature = value;
        } else {
            throw new IllegalArgumentException("Sıcaklık değeri kabul
edilen aralıkta değil: " +
                    MIN_TEMPERATURE + " ile " + MAX_TEMPERATURE + "
arasında.");
        } }
    // Alt bölmenin sıcaklığı
    public double getLowerCompartmentTemperature() {
        return lowerCompartmentTemperature;
    }
    public void setLowerCompartmentTemperature(double value) {
        if (value >= MIN_TEMPERATURE && value <= MAX_TEMPERATURE) {
            this.lowerCompartmentTemperature = value;
        } else {
            throw new IllegalArgumentException("Sıcaklık değeri kabul
edilen aralıkta değil: " +
                    MIN_TEMPERATURE + " ile " + MAX_TEMPERATURE + "
arasında.");
        }}
    // Ortalama sıcaklık
    public double getAverageTemperature() {
        return (upperCompartmentTemperature +
lowerCompartmentTemperature) / 2.0;
    }}
```

DoorLock sınıfı, akıllı kapı kilidi cihazlarının durumunu ve güvenliğini yönetmek için tasarlanmıştır. Bu sınıf, SmartDevice sınıfından türetilmiş olup temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, kapıyı açıp kapatabilir ve 4 haneli bir şifre belirleyebilir. Şifre doğrulama işlemi, regex kullanılarak güvence altına alınmıştır. Kod, kullanıcı hatalarını önlemek ve cihazın güvenli bir şekilde çalışmasını sağlamak için hata yönetimi mekanizmalarıyla donatılmıştır.

```
package Devices;
public class DoorLock extends SmartDevice {
    private String password; // Şifre 4 haneli olarak tutulur (örn.
"1234")

    // Yapıcı metod
    public DoorLock(String name, boolean status) {
        super(name, status);
    }
    // Durum gösterir
    public boolean isRunning() {return super.getStatus();}
    // Kapıyı açar
    @Override
    public void turnOn() {
        try {
            super.turnOn(); // Üst sınıfın turnOn metodu
        } catch (Exception ignored) {
            // Hata formda işlenmek üzere bırakılır, burada işlenmez}
    }
    // Kapıyı kapatır
    @Override
    public void turnOff() {
        try {
            super.turnOff(); // Üst sınıfın turnOff metodu
        } catch (Exception ignored) {
            // Hata formda işlenmek üzere bırakılır
        }
    }
    // Şifre belirleme (4 haneli validasyon yapılır)
    public boolean setPassword(String input) {
        try {
            if (input != null && input.matches("\\d{4}")) { // Şifre 4
haneli mi kontrol et
                this.password = input;
                return true; // Başarılı}
        } catch (Exception ignored) {
            // Hata formda işlenmek üzere bırakılır
        }
        return false; // Hatalı
    }
    // Şifreyi alma (opsiyonel, gizlilik açısından kullanılmayabilir)
    public String getPassword() {
        try {
            return this.password;
        } catch (Exception ignored) {
            // Hata formda işlenmek üzere bırakılır
        }
        return null; // Hata durumunda null döner
    }}
```

Dishwasher sınıfı, akıllı bulaşık makinelerinin çalışma durumunu ve programlarını yönetmek için tasarlanmıştır. SmartDevice sınıfından türetilmiş olan bu sınıf, temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, cihazı açıp kapatabilir, farklı yıkama programlarını seçebilir ve aktif programın durumunu izleyebilir. Sınıf, program adı ve süresini bir HashMap yapısında saklar, böylece programlara kolay erişim ve yönetim sağlar. Kod, kullanıcıya esneklik sunmak ve cihazın güvenli şekilde çalışmasını sağlamak amacıyla uygun kontrol mekanizmalarıyla yapılandırılmıştır.

```
package Devices;
import java.util.HashMap;
import java.util.Map;
public class Dishwasher extends SmartDevice {
    private boolean isRunning; // Makine çalışıyor mu?
    private String currentProgram; // Aktif program
    private final Map<String, Integer> programs; // Programlar ve
süreleri
    public Dishwasher(String name, boolean status) {
        super(name, status);
        this.isRunning = status;
        this.currentProgram = null;
        // Programları oluştur
        this.programs = new HashMap<>();
        programs.put("Hızlı Yıkama", 30);
        programs.put("Ekonomik Yıkama", 60);
        programs.put("Yoğun Yıkama", 90);
        programs.put("Günlük Yıkama", 45);
        programs.put("Hassas Yıkama", 75);
    }
    // Cihazı aç
    public void turnOn() {
        isRunning = true;}
    // Cihazı kapat
    public void turnOff() {
        isRunning = false;
        currentProgram = null; // Program sıfırla}
    // Program seç
    public void selectProgram(String programName) {
        if (programs.containsKey(programName)) {
            currentProgram = programName;
        }}
    // Çalışma durumu
    public boolean isRunning() {
        return isRunning;}
    // Şu anki program
    public String getCurrentProgram() {
        return currentProgram;}
    // Programlar (Map olarak döndür)
    public Map<String, Integer> getPrograms() {
        return programs;
    }}
```

RobotVacuum sınıfı, akıllı robot süpürgelerinin çalışma durumunu ve oda seçimini yönetmek için tasarlanmıştır.SmartDevice sınıfından türetilmiş olan bu sınıf, temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, cihazı açıp kapatabilir, farklı odaları seçebilir ve aktif odanın durumunu izleyebilir. Sınıf, oda kodları ve adlarını bir HashMap yapısında saklar, böylece odalar arasında kolay geçiş ve yönetim sağlar. Kod, kullanıcıya esneklik sunmak ve robot süpürgenin güvenli şekilde çalışmasını sağlamak amacıyla uygun kontrol mekanizmalarıyla yapılandırılmıştır.

```
package Devices;
import java.util.Map;
import java.util.HashMap;
public class RobotVacuum extends SmartDevice implements RobotVacuumInterface {
    private boolean isRunning; // Süpürgenin çalışıp çalışmadığını belirtir
    private String selectedRoom; // Seçilen odanın adını tutar
    private final Map<String, String> roomOptions; // Mevcut odaların kodları ve
adları
    // Constructor
    public RobotVacuum(String name, boolean status) {
        super(name, status);
        this.isRunning = status;
        this.selectedRoom = null;
        // Oda seçeneklerini başlat
        this.roomOptions = new HashMap<>();
        this.roomOptions.put("L", "Oturma Odası");
        this.roomOptions.put("K", "Mutfak");
        this.roomOptions.put("B", "Yatak Odası");
        this.roomOptions.put("Y", "Çalışma Odası");}
    @Override
    public void startCleaning() throws CustomRobotVacuumException {
        if (isRunning) {
            throw new CustomRobotVacuumException("Süpürge zaten çalışıyor.");}
        if (selectedRoom == null) {
            throw new CustomRobotVacuumException("Temizlenecek bir oda
seçilmedi!");}
        isRunning = true; // Süpürge çalışmaya başla}
    @Override
    public void stopCleaning() {
        if (isRunning) {
            isRunning = false; // Süpürge durdurulur
        }}
    @Override
    public Map<String, String> listRooms() {
        return new HashMap<>(roomOptions); // Mevcut oda listesini döndürür}
    @Override
    public boolean selectRoom(String roomCode) throws CustomRobotVacuumException
{
        if (!roomOptions.containsKey(roomCode)) {
            throw new CustomRobotVacuumException("Geçersiz oda seçimi!");}
        if (isRunning) {
            throw new CustomRobotVacuumException("Süpürge çalışırken oda
değiştirilemez.");}
        this.selectedRoom = roomOptions.get(roomCode); // Oda seçilir
        return true;}
    @Override
    public boolean isVacuumRunning() {
        return isRunning; // Süpürgenin çalışıp çalışmadığını döndürür}
    @Override
    public String getSelectedRoom() {
        return selectedRoom; // Seçilen odanın adını döndürür
    }}
```

SmartCurtain sınıfı, akıllı perdelerin çalışma durumunu ve açıklık yüzdesini yönetmek için tasarlanmıştır. SmartDevice sınıfından türetilmiş olan bu sınıf, temel cihaz kontrol işlevlerini devralır ve özelleştirir. Kullanıcı, cihazı açıp kapatabilir ve perdenin açıklık yüzdesini ayarlayarak ne kadar açıldığını kontrol edebilir. Sınıf, perde açıklığını bir tam sayı olarak tutar, böylece kullanıcı perdeyi istediği şekilde ayarlayabilir. Kod, kullanıcıya esneklik sunmak ve cihazın güvenli şekilde çalışmasını sağlamak amacıyla uygun kontrol mekanizmalarıyla yapılandırılmıştır. SmartCurtain, temel açma ve kapama işlemlerini SmartDevice sınıfından alırken, perde açıklığını belirlemek ve mevcut durumu öğrenmek için özel metodlar sunar. Bu metodlar, kullanıcının perdenin açılma seviyesini hassas bir şekilde kontrol etmesine olanak tanır. Ayrıca, açıklık yüzdesinin geçerli bir aralıkta olup olmadığını denetleyen hata kontrol mekanizmaları içerir.

```
package Devices;

public class SmartCurtain extends SmartDevice {

    // Perdenin açıklık yüzdesini tutan değişken
    private int openPercentage;

    // Constructor (yapıcı metod)
    public SmartCurtain(String name, boolean status) {
        super(name, status); // Üst sınıfın constructor'ını çağırır.
        this.openPercentage = 0; // Başlangıçta perde tamamen kapalıdır.
    }

    // Perde sistemini açar
    @Override
    public void turnOn() {
        super.turnOn();
    }

    // Perde sistemini kapatır
    @Override
    public void turnOff() {
        super.turnOff();
    }

    // Perde açıklık yüzdesini ayarlar
    public void setOpenPercentage(int percentage) {
        if (percentage >= 0 && percentage <= 100) {
            this.openPercentage = percentage;
        } else {
            throw new IllegalArgumentException("Açıklık yüzdesi 0 ile
100 arasında olmalıdır.");
        }
    }

    // Perdenin mevcut açıklık yüzdesini döndürür
    public int getOpenPercentage() {
        return this.openPercentage;
    }
}
```

Television sınıfı, akıllı televizyonların çalışma durumunu ve temel fonksiyonlarını yönetmek için tasarlanmıştır. SmartDevice sınıfından türetilmiş olan bu sınıf, televizyonu açıp kapatabilme, kanal değiştirebilme ve ses seviyesini ayarlayabilme gibi işlevler sunar. Sınıf, kanal listesini ve mevcut kanalı bir liste içinde tutar, böylece kullanıcı istediği kanalı kolayca seçebilir. Ses seviyesi ise 0 ile 100 arasında bir tam sayı olarak ayarlanabilir. Kod, kullanıcıya esneklik sunmak ve televizyonun güvenli şekilde çalışmasını sağlamak amacıyla uygun hata kontrol mekanizmalarıyla yapılandırılmıştır. Television sınıfı, televizyonun açılıp kapanması, kanal değiştirilmesi ve ses seviyesinin ayarlanması gibi temel işlevleri yönetirken, kanal listesini de dinamik bir şekilde sağlar. Kullanıcı, kanal index numarasını kullanarak kanalları değiştirebilir ve ses seviyesini geçerli bir aralıkta tutarak cihazı güvenli bir şekilde kullanabilir.

```
package Devices;

import java.util.ArrayList;
import java.util.List;

public class Television extends SmartDevice {
    private boolean isOn; // Televizyonun açık/kapalı durumu
    private final List<String> channelList; // Kanal listesi
    private String currentChannel; // Şu anki kanal (aktif kanal)
    private int volumeLevel; // Ses seviyesi (0-100 arasında)

    // Constructor: Yeni bir televizyon objesi oluşturur
    public Television(String name, boolean status) {
        super(name, status);
        this.isOn = status;
        this.currentChannel = null; // Varsayılan olarak hiç bir kanal ayarlanmaz
        this.volumeLevel = 50; // Varsayılan ses seviyesi

        // Kanal listesini oluştur
        this.channelList = new ArrayList<>();
        channelList.add("TRT 1");
        channelList.add("ATV");
        channelList.add("Show TV");
        channelList.add("Kanal D");
        channelList.add("Star TV");
        channelList.add("FOX TV");
        channelList.add("TV8");
        channelList.add("Kanal 7");
        channelList.add("NTV");
        channelList.add("HaberTürk");
    }

    // GETTER VE SETTER METOTLAR:

    // Televizyonun açık/kapalı durumunu kontrol et
    public boolean isOn() {
        return this.isOn;
    }

    // Mevcut ses seviyesini al
    public int getVolumeLevel() {
        return volumeLevel;
    }

    // Mevcut seçili kanal
    public String getCurrentChannel() {
        return currentChannel == null ? "Kanal seçilmedi" : currentChannel;
    }

    // Tüm kanal listesini al
    public List<String> getChannelList() {
        return channelList;
    }

    // TELEVİZYONUN TEMEL FONKSİYONLARI:

    // Televizyonu açar
    public void turnOn() {
        this.isOn = true;
    }

    // Televizyonu kapatır
    public void turnOff() {
        this.isOn = false;
        this.currentChannel = null; // Kapandığında kanal sıfırlanır
    }


    // Ses seviyesini belirler
    public void setVolume(int volume) {
        if (volume >= 0 && volume <= 100) {
            this.volumeLevel = volume;
        } else {
            throw new IllegalArgumentException("Ses seviyesi 0-100 arasında olmalıdır.");
        }
    }

    // Kanal değiştirir
    public void changeChannel(int channelIndex) {
        if (channelIndex >= 0 && channelIndex < channelList.size()) {
            this.currentChannel = channelList.get(channelIndex);
        } else {
            throw new IllegalArgumentException("Geçersiz kanal indexi.");
        }
    }
}
```

Thermostat sınıfı, akıllı termostatların çalışma durumunu ve sıcaklık ayarını yönetmek için tasarlanmıştır. SmartDevice sınıfından türetilmiş olan bu sınıf, termostatın açılıp kapatılmasını sağlar ve mevcut sıcaklık değerini ayarlayarak kontrol eder. Sınıf, geçerli sıcaklık değerini 15 ile 25 derece arasında tutarak kullanıcının sıcaklık ayarlarını güvenli bir şekilde yapmasını sağlar. Kod, kullanıcıya esneklik sunmak ve cihazın güvenli şekilde çalışmasını sağlamak amacıyla uygun hata kontrol mekanizmalarıyla yapılandırılmıştır. Thermostat sınıfı, cihazın açılması ve kapatılması işlemlerini SmartDevice sınıfından devir alırken, sıcaklık ayarını kontrol eden ve geçerli aralık dışı değerler için hata mesajı veren özel metodlar sunar. Bu, kullanıcıların cihazı güvenli ve verimli bir şekilde kullanmalarına olanak tanır.

```
package Devices;

public class Thermostat extends SmartDevice {

    private double currentTemperature = 20.0; // Varsayılan sıcaklık

    public Thermostat(String name, boolean status) {
        super(name, status);
    }

    public void turnOn() {
        super.turnOn(); // Sadece cihazın açık/kapalı durumunu
değiştirir
    }

    public void turnOff() {
        super.turnOff(); // Sadece cihazın açık/kapalı durumunu
değiştirir
    }

    public void setTemperature(double temperature) {
        if (temperature < 15.0 || temperature > 25.0) {
            throw new IllegalArgumentException("Geçersiz sıcaklık
aralığı! 15 ile 25 arasında bir değer olmalıdır.");
        }
        this.currentTemperature = temperature;
    }

    public double getCurrentTemperature() {
        return currentTemperature;
    }
}
```

WashingMachine sınıfı, akıllı çamaşır makinelerinin çalışma durumunu ve program seçimlerini yönetmek için tasarlanmıştır. SmartDevice sınıfından türetilmiş olan bu sınıf, çamaşır makinesinin açılmasını, kapatılmasını ve farklı yıkama programlarının seçilmesini sağlar. Sınıf, her programın kodunu ve açıklamasını tutan bir HashMap yapısı kullanarak, kullanıcıların kolayca program seçmelerini sağlar. Çamaşır makinesi, bir program seçildikten sonra başlatılabilir ve durdurulabilir. Kod, kullanıcıya esneklik sunmak ve cihazın güvenli şekilde çalışmasını sağlamak amacıyla uygun hata kontrol mekanizmalarıyla yapılandırılmıştır. WashingMachine sınıfı, çamaşır makinesinin çalışıp çalışmadığını kontrol etmek, seçilen programı görmek ve geçerli bir program kodu ile program seçimi yapmak için çeşitli metotlar sunar. Ayrıca, makineyi başlatma ve durdurma işlevleri de kullanıcıya kontrol sağlar. Makinenin çalışma süreci tamamlandığında, işlem otomatik olarak sonlandırılır.

```
package Devices;

import java.util.HashMap;
import java.util.Map;

public class WashingMachine extends SmartDevice {

    private boolean isRunning; // Makine çalışıyor mu?
    private String selectedProgram; // Seçilen program kodu
    private final Map<String, String> programDescription; // Program kodları ve açıklamaları

    // Constructor
    public WashingMachine(String name, boolean status) {
        super(name, status);
        this.isRunning = status;
        this.selectedProgram = null;
        this.programDescription = new HashMap<>();

        // Program kodları ve açıklamaları
        this.programDescription.put("A", "Ön yıkamalı pamuklu (50°-95°)");
        this.programDescription.put("D", "Ön yıkamalı sentetik (30°-60°)");
        this.programDescription.put("B", "Pamuklu (30°-95°)");
        this.programDescription.put("F", "Sentetik (30°-60°)");
        this.programDescription.put("X", "Pamuklu kısa (30°-50°)");
        this.programDescription.put("G", "Sentetik kısa (Soğuk-30°)");
        this.programDescription.put("M", "Pamuklu renkli (Soğuk-40°)");
        this.programDescription.put("J", "Yünlü (Soğuk-40°)");
        this.programDescription.put("C", "Pamuklu sıkma");
        this.programDescription.put("H", "Sentetik/Yünlü sıkma");
    }

    // Çamaşır makinesinin çalışıp çalışmadığını kontrol eder
    public boolean isRunning() {
        return isRunning;
    }

    // Seçilen programı döner
    public String getSelectedProgram() {
        return selectedProgram;
    }

    // Tüm programları döner
    public Map<String, String> getProgramDescription() {
        return programDescription;
    }

    // Program seçimi ve doğrulama
    public boolean setProgram(String programCode) {
        if (programDescription.containsKey(programCode)) {
            selectedProgram = programCode;
            return true; // Geçerli program seçildi
        }
        return false; // Hatalı program kodu
    }

    // Makineyi başlatır
    public boolean start() {
        if (!isRunning && selectedProgram != null) {
            isRunning = true;
            return true; // Başlatıldı
        }
        return false; // Zaten çalışıyor ya da program seçilmemiş
    }

    // Makineyi durdurur
    public boolean stop() {
        if (isRunning) {
            isRunning = false;
            return true; // Durduruldu
        }
        return false; // Zaten durmuş
    }

    // Yıkama işlemini simüle eden bir metot (örnek bir bekleme mantığı eklenebilir)
    public void completeProgram() {
        if (isRunning) {
            // İşlem bittiğinde durdurur
            stop();
        }
    }
}
```

3. Sonuç ve Değerlendirme

Projenin sonunda:

Proje, akıllı cihazların merkezi bir platform üzerinden yönetilmesini sağlayarak kullanıcıların günlük yaşamlarını kolaylaştırmayı amaçladı. Geliştirilen sistem, farklı marka ve modeldeki cihazları bir araya getirerek tüm yönetim işlemlerini tek bir noktadan yapılabilir hale getirdi. Bu entegrasyon, kullanıcıların cihazları kontrol etmek için birden fazla uygulama kullanma ihtiyacını ortadan kaldırarak zaman ve çaba tasarrufu sağladı.

Kullanıcı dostu bir grafiksel arayüz (GUI) ile cihazların kontrolü, karmaşık ayarların ve işlemlerin bile kolayca yönetilmesini mümkün kıldı. Kullanıcıların cihazlarla etkileşime girmesi oldukça basit hale gelirken, aynı zamanda arayüz tasarımında estetik ve işlevsellik dengesine de dikkat edildi. Sistem, kullanıcıların her cihazı özelleştirilmiş ayarlarla kontrol etmelerine olanak tanıdı; bu da kullanıcı deneyimini daha verimli ve keyifli bir hale getirdi.

Projenin geliştirilmesi sırasında bazı teknik zorluklarla karşılaşıldı. Özellikle cihazların farklı protokollerle çalışıyor olması, başlangıçta sistemin entegrasyonunu zorlaştırdı. Ancak, bu sorunlar tasarım mimarisinde yapılan iyileştirmeler ve modüler yapı kullanılarak çözüme kavuşturuldu. Kod yapısının esnek ve genişletilebilir olması, ilerleyen zamanlarda yeni cihazların eklenmesini kolaylaştıracak şekilde tasarlandı.

Sonuç olarak, SmartDeviceSystem, akıllı ev cihazlarını yönetmenin geleneksel yollarını bir adım ileriye taşıyan, kullanıcı dostu ve verimli bir çözüm sundu. Sistem, pratikteki tüm gereksinimleri karşılayacak şekilde geliştirildi ve gelecekte daha fazla cihazın entegre edilmesi için sağlam bir altyapıya sahip.

4. Ekler

```
import javax.swing.*;

public class Main {

    public static void main(String[] args) {

        // burada uygulamanın çalışması için JFrame Class'ından bir
nesne oluşturup aşağıda çalıştırdık.
        //kodumuz loginFormdan başladığı için new loginform açarak
parentFrame nesnesini içine attık.

        JFrame parentFrame = new JFrame();
        new LoginForm(parentFrame);

    }
}
```

Şekil 2: Main Çalıştırma Kodu

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import java.awt.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;

// Cihaz sınıflarını import edelim
import DeviceForms.*;
import DeviceForms.RobotVacuumForm.RobotVacuumForm;
import Devices.*;
```

Şekil 3: Main Menu Kodu

```
public class MainMenuForm {
    //Mainform classında kullanılan butonlar ve değişkenlerin
tanımlanması

    private JButton addDeviceButton;
    private JButton removeDeviceButton;
    private JTable table1;
    private JPanel mainPanel;
    private JLabel lblDateTime;

    private final User currentUser; // Şu anda giriş yapan kullanıcı

    public MainMenuForm(User user) {
        this.currentUser = user;
        //formun düzenlenmesi
        JFrame frame = new JFrame("Ana Menü");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(mainPanel);
        frame.setSize(600, 400);
        frame.setVisible(true);

        // üstte gözükecek tarih ve zaman

        Timer timer = new Timer(1000, e -> {
            LocalDateTime now = LocalDateTime.now();
            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
            lblDateTime.setText(now.format(formatter));
        });
        timer.start();


        // Kullanıcının cihazlarını JTable'a dolduralması
        updateDeviceTable();

        // Cihaz ekleme butonuna aksiyon ekleyelim
        addDeviceButton.addActionListener(e -> handleAddDevice());
        // Cihaz kaldırma butonuna aksiyon ekleyelim
        removeDeviceButton.addActionListener(e -> handleRemoveDevice());
    }

    // Tabloyu cihazlarla doldur
    private void updateDeviceTable() {
        String[] columnNames = {"Cihaz Adı", "İşlemler"}; // Veritabanı
sütun isimleri
        List<SmartDevice> devices = currentUser.getDevices();

        Object[][] rowData = new Object[devices.size()][2];

        int i = 0;
        for (SmartDevice device : devices) {
            rowData[i][0] = device.getName(); // Cihaz adı
            rowData[i][1] = "İşlemler"; // İşlemler butonu
            i++; }
```

Şekil 3.1: Main Menu Kodu

```
        table1.setModel(new DefaultTableModel(rowData, columnNames) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return column == 1; // Sadece "İşlemler" hücresini
düzenlenebilir yapıyoruz (buton)
            }
        });

        // İşlemler sütunu için özel renderer ve editor
        table1.getColumn("İşlemler").setCellRenderer(new
ButtonRenderer());
        table1.getColumn("İşlemler").setCellEditor(new ButtonEditor(new
JCheckBox()));
    }

    // "Cihaz Ekle" butonuna tıklanınca
    private void handleAddDevice() {
        String[] availableDevices = {"Light", "Dishwasher",
"Thermostat", "DoorLock", "Fridge", "WashingMachine", "SmartCurtain",
"Television", "RobotVacuum"};

        String selectedDeviceType = (String)
JOptionPane.showInputDialog(
                null,
                "Cihaz Türü Seçiniz:",
                "Cihaz Ekleme",
                JOptionPane.QUESTION_MESSAGE,
                null,
                availableDevices,
                availableDevices[0]
        );

        if (selectedDeviceType != null) {
            String deviceName = JOptionPane.showInputDialog("Cihaz
İsmini Giriniz:");
            if (deviceName != null && !deviceName.isBlank()) {
                SmartDevice newDevice = createDevice(selectedDeviceType,
deviceName);

                if (newDevice != null) {
                    currentUser.addDevice(newDevice);
                    updateDeviceTable(); // Tabloyu yeniden güncelle
                }
            } else {
                JOptionPane.showMessageDialog(null, "Geçersiz cihaz
adı!", "Hata", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

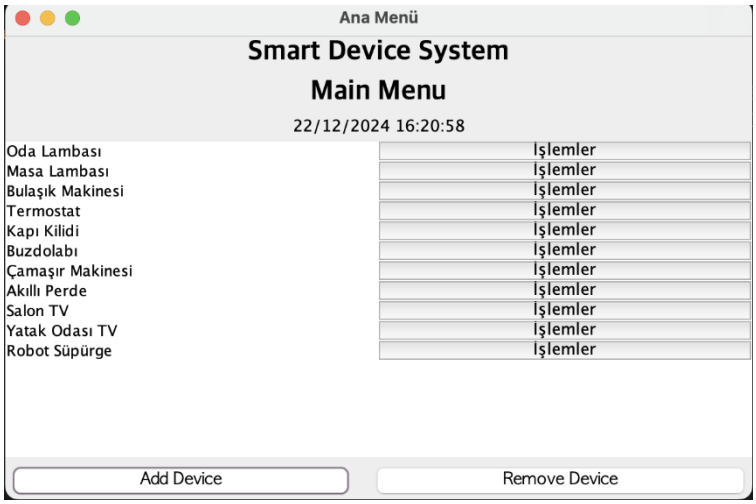
Şekil 3.2: Main Menu Kodu

```
    // "Cihaz Kaldır" butonuna tıklanınca
    private void handleRemoveDevice() {
        int selectedRow = table1.getSelectedRow();
        if (selectedRow >= 0) {
            String deviceName = table1.getValueAt(selectedRow, 0).toString();
            SmartDevice deviceToRemove = currentUser.getDevices().stream()
                    .filter(device -> device.getName().equals(deviceName))
                    .findFirst()
                    .orElse(null);
            if (deviceToRemove != null) {
                currentUser.removeDevice(deviceToRemove);
                updateDeviceTable(); // Tabloyu güncelle}
        } else {
            JOptionPane.showMessageDialog(null, "Kaldırmak için bir cihaz
seçmelisiniz.", "Uyarı", JOptionPane.WARNING_MESSAGE);
        }
    }
    // İşlemler butonu için cihaz türüne göre form aç
    private void handleDeviceDetails(int row) {
        String deviceName = table1.getValueAt(row, 0).toString();
        SmartDevice selectedDevice = currentUser.getDevices().stream()
                .filter(device -> device.getName().equals(deviceName))
                .findFirst()
                .orElse(null);
        if (selectedDevice != null) {
            if (selectedDevice instanceof Light) {
                new LightForm((Light) selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof Dishwasher) {
                new DishwasherForm((Dishwasher)
selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof Thermostat) {
                new ThermostatForm((Thermostat)
selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof DoorLock) {
                new DoorLockForm((DoorLock) selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof Fridge) {
                new FridgeForm((Fridge) selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof WashingMachine) {
                new WashingMachineForm((WashingMachine)
selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof SmartCurtain) {
                new SmartCurtainForm((SmartCurtain)
selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof Television) {
                new TelevisionForm((Television)
selectedDevice).setVisible(true);
            } else if (selectedDevice instanceof RobotVacuum) {
                new RobotVacuumForm((RobotVacuum)
selectedDevice).setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Bu cihaza özel bir işlem
bulunamadı.", "Bilinmeyen Cihaz", JOptionPane.WARNING_MESSAGE);
            }
        }
    }
```

Şekil 3.3: Main Menu Kodu

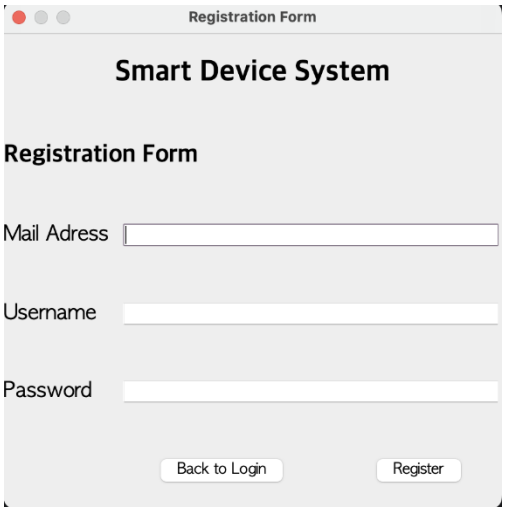
```
    // Dinamik cihaz nesnesi oluştur
    private SmartDevice createDevice(String type, String name) {
        switch (type) {
            case "Light":
                return new Light(name, false);
            case "Dishwasher":
                return new Dishwasher(name, false);
            case "Thermostat":
                return new Thermostat(name, false);
            case "DoorLock":
                return new DoorLock(name, false);
            case "Fridge":
                return new Fridge(name, false);
            case "WashingMachine":
                return new WashingMachine(name, false);
            case "SmartCurtain":
                return new SmartCurtain(name, false);
            case "Television":
                return new Television(name, false);
            case "RobotVacuum":
                return new RobotVacuum(name, false);
            default:
                return null;}
    }
    // Tabloya buton eklemek için renderer
    class ButtonRenderer extends JButton implements TableCellRenderer {
        public ButtonRenderer() {
            setOpaque(true);}
        @Override
        public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int column) {
            setText((value == null) ? "" : value.toString());
            return this;}
    }
    // Tabloya buton işlevsellik kazandırmak için editor
    class ButtonEditor extends DefaultCellEditor {
        private final JButton button = new JButton();
        private int row;

        public ButtonEditor(JCheckBox checkBox) {
            super(checkBox);
            button.setOpaque(true);
            button.addActionListener(e -> {
                fireEditingStopped(); // Düzenlemeyi sonlandır
                handleDeviceDetails(row); // Satıra göre işlem yap
            });}
        @Override
        public Component getTableCellEditorComponent(JTable table, Object value,
boolean isSelected, int row, int column) {
            this.row = row; // Hangi satırda tıklanıldığını al
            button.setText((value == null) ? "" : value.toString());
            return button; }
        @Override
        public Object getCellEditorValue() {
            return button.getText();
        }
    }
}
```

Şekil 3.4: Main Menu Kodu



Şekil 4: Main Menu Ekranı

Şekil 5: Login (Giriş) Ekranı



Şekil 6: Registration (Kayıt) Ekranı

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

public class LoginForm extends JDialog {

    //Loginform classında kullanılan butonlar ve değişkenlerin tanımlanması

    private JTextField tfUsername;
    private JPasswordField tfPassword;
    private JButton loginButton;
    private JButton createAnAccountButton;
    private JPanel signPanel;

    // Kullanıcı bilgilerini içeren dosya

    private static final File USER_FILE = new File("users.txt");

    public LoginForm(JFrame parent) {
        super(parent);
        setTitle("Login Form");
        setContentPane(signPanel);
        setLocationRelativeTo(parent);
        setModal(true);
        setMinimumSize(new Dimension(450, 450));
        setResizable(false);

        // Login butonuna tıklama işlemi

        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = tfUsername.getText(); // Kullanıcı adını al
                String password = new String(tfPassword.getPassword()); //
Şifreyi al

                if (username.isEmpty() || password.isEmpty()) {
                    JOptionPane.showMessageDialog(LoginForm.this, "Tüm alanlar
doldurulmalıdır!", "Hata", JOptionPane.ERROR_MESSAGE);
                    return;
                }
```

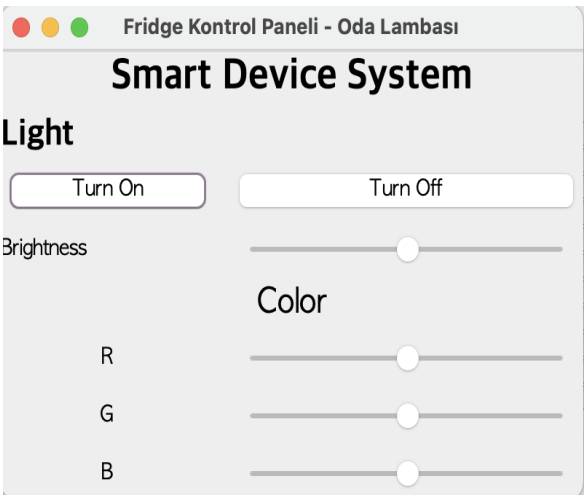
Şekil 7.1: LoginForm (Giriş Ekranı) Kodu

```
try {
                    // Kullanıcı doğrulaması için authenticateFromFile çağrısı

                    boolean isAuthenticated =
User.authenticateFromFile(username, password, USER_FILE);
                    if (isAuthenticated) {
                        JOptionPane.showMessageDialog(LoginForm.this, "Giriş
Başarılı!", "Başarı", JOptionPane.INFORMATION_MESSAGE);
                        dispose();
                        // Başarılı giriş sonrası ana menüye yönlendirme
                        MainMenuForm mainMenuForm = new MainMenuForm(new
User(username, password)); // Örnek (Ana Menü Ekranı)
                    } else {
                        JOptionPane.showMessageDialog(LoginForm.this, "Hatalı
kullanıcı adı veya şifre!", "Hata", JOptionPane.ERROR_MESSAGE);
                    }
                } catch (IOException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(LoginForm.this, "Dosya okuma
sırasında bir hata oluştu.", "Hata", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        // Hesap oluşturma butonuna tıklama işlemi
        createAnAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose(); // Login ekranını kapat
                new RegistrationForm(parent); // Kayıt ekranını aç
            }
        });

        setVisible(true);
        pack();
    }
}
```

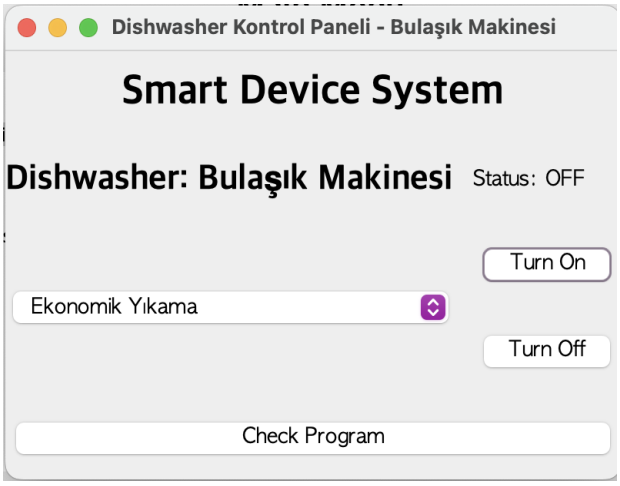
Şekil 7.2: LoginForm (Giriş Ekranı) Kodu

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
public class RegistrationForm extends JDialog {
    private JButton registerButton;
    private JPasswordField createPasswordField;
    private JTextField createUsernameField;
    private JTextField createMailAdressField;
    private JPanel registrationPanel;
    private JButton backToLoginButton;
    private static final File USER_FILE = new File("users.txt"); // Kullanıcı
bilgilerini kaydedeceğimiz dosya
    public RegistrationForm(JFrame parent) {
        super(parent);
        setTitle("Registration Form");
        setContentPane(registrationPanel);
        setLocationRelativeTo(parent);
        setModal(true);
        setMinimumSize(new Dimension(450, 450));
        setResizable(false);
        // Kayıt butonuna tıklama olayı
        registerButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String mailAdress = createMailAdressField.getText();
                String username = createUsernameField.getText();
                String password = new String(createPasswordField.getPassword());
                if (username.isEmpty() || password.isEmpty() ||
mailAdress.isEmpty()) {
                    JOptionPane.showMessageDialog(RegistrationForm.this, "Tüm
alanlar doldurulmalıdır!", "Hata", JOptionPane.ERROR_MESSAGE);
                    return;}
                User newUser = new User(username, password);
                try {
                    newUser.saveToFile(USER_FILE); // Yeni kullanıcıyı dosyaya
kaydet
                    JOptionPane.showMessageDialog(RegistrationForm.this, "Kayıt
Başarılı!", "Başarı", JOptionPane.INFORMATION_MESSAGE);
                    dispose();
                    new LoginForm(parent); // Giriş ekranına yönlendir
                } catch (IOException ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(RegistrationForm.this, "Kayıt
sırasında bir hata oluştu.", "Hata", JOptionPane.ERROR_MESSAGE);
                }
            }
        });
        // Geri butonuna tıklama olayı
        backToLoginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                dispose();
                new LoginForm(parent); // Giriş ekranına dön
            }
        });
        pack();
        setVisible(true);
    }}
```

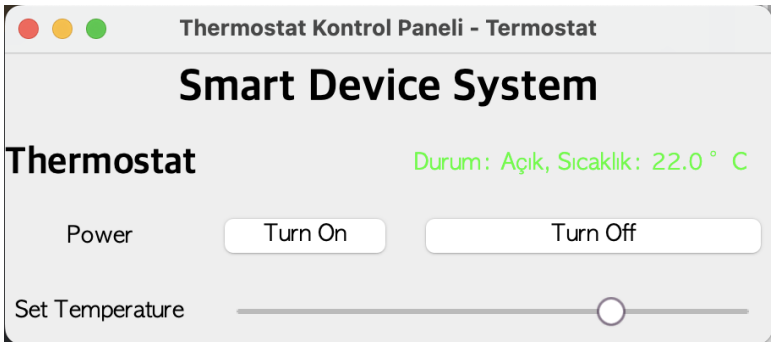
Şekil 8: RegistrationForm (Kayıt Ekranı) Kodu



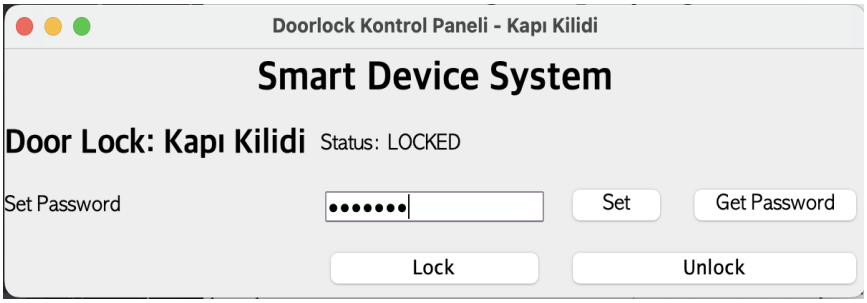
Şekil 9: Fridge (Buzdolabı) Ekranı



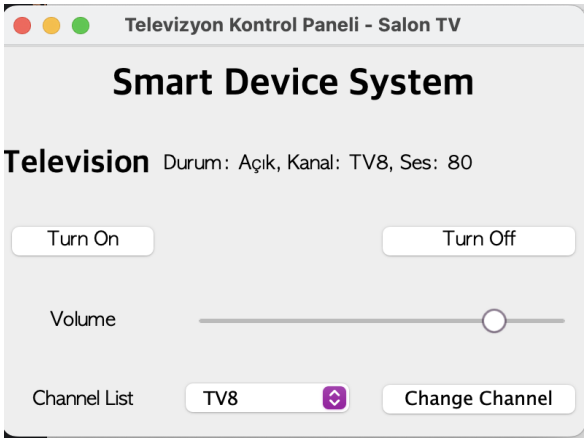
Şekil 10: Dishwasher (Bulaşık Makinesi) Ekranı



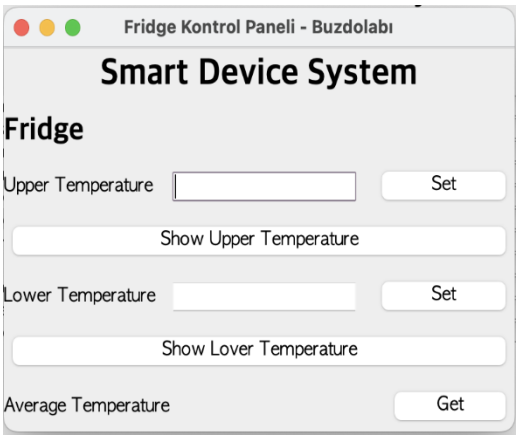
Şekil 11: Thermostat (Termostat) Ekranı



Şekil 12: Thermostat (Termostat) Ekranı

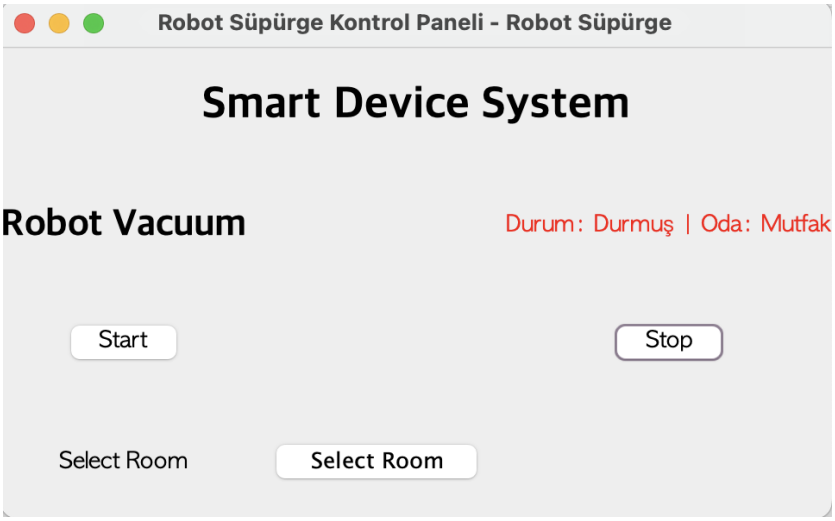


Şekil 13: TV (Televizyon) Ekranı

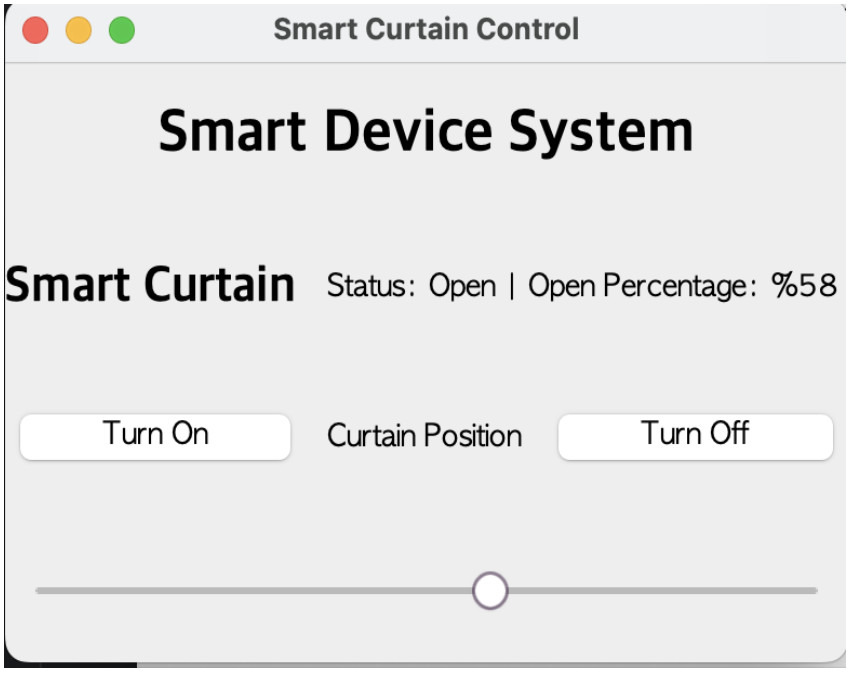


Şekil 14: Fridge (Buzdolabı) Ekranı

Şekil 15: Washing Machine (Çamaşır Makinesi) Ekranı



Şekil 16: Robot Vacuum (Robot Süpürge) Ekranı



Şekil 17: Smart Curtain (Akıllı Perde) Ekranı

5. Kaynakça

[1] R. K. Gupta, "Smart Home Automation: A Survey," *International Journal of Computer Applications*, vol. 167, no. 7, pp. 38-44, Jan. 2017.
[2] M. B. Garcia, A. Martinez, and F. Ruiz, "IoT-based Smart Home Systems," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 10-19, Feb. 2018.
[3] J. Smith, "Designing Home Automation Systems Using Java and IoT," *Proceedings of the International Conference on Smart Systems and IoT*, pp. 102-108, 2019.
[4] D. Kumar, P. Shah, and R. Mehta, "Building a Smart Device System for Home Automation with Java Swing," *Journal of Software Engineering and Applications*, vol. 11, no. 3, pp. 123-130, March 2020.
[5] S. P. Koirala, *Building Smart Homes with Java: A Practical Guide*, 2nd ed., New York, NY: Wiley, 2018.
[6] A. L. Johnson and H. H. Thompson, "Advancements in Smart Home Technologies," *Advanced Technology Review*, vol. 25, pp. 45-60, 2022.