

**İSTANBUL TOPKAPI ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**YAZILIM MÜHENDİSLİĞİ**

**2022 -2023 / Bahar Dönemi**

**Yazılım Mimarisi Vize Ödevi**

**Mehmet Akif**

**Küçükkaya**

**21040301063**

**Teslim Tarihi: 17.04.2023**

Yazılım Mimarisi Vize Ödevi

# Senaryo:

Bir otopark işletmecisi şirketinize başvurarak sizden işletmesi için bir yazılım geliştirmenizi, tüm otopark operasyonlarını bu yazılım üzerinden takip etmek istediğini iletmiştir.

Müşteri ile yaptığınız görüşmeler sonucu şu bilgileri edindiniz.

* Müşterinin biri açık, diğeri kapalı olmak üzere iki farklı otoparkı bulunmaktadır
* Her bir otopark için, birer güvenlik ve birer park görevlisi otoparkta görev yapmaktadır. Ayrıca tüm finansal işlemlerden sorumlu bir muhasebeci, tüm çalışanlardan sorumlu bir müdür ve işletme sahibi olarak da kendisi bulunmaktadır.
* Kapalı otopark maksimum 100, açık otopark 50 adet araç kapasitelidir.
* Müşteriler sadece kapalı otopark için aylık ve yıllık olabilecek abonelik kaydı yaptırabilmektedirler. Standart binek türü araçlar için aylık 100 TL, yıllık 1000 TL otopark ücreti bulunmaktadır. Daha büyük araçlar için ise 150 TL aylık, 1500 TL ise yıllık ücret geçerlidir.
* Otoparklara girişler güvenli bariyer sistemi ile yapılmakta abone olmayan müşteriler girişte otopark bileti alıp çıkışta ibraz ederek ücret ödemektedirler.
* Aboneler için sticker uygulaması vardır.

Müşteri ve proje ekibi ile yaptığınız diğer görüşmelerde şu gereksinimleri not ettiniz;

* Otopark operasyonları için geliştirilecek yazılım web tabanlı ve mobil tabanlı olacaktır.
* Hem çalışanlar hem de müşteriler bu yazılıma kendi kullanıcıları ile giriş yapıp rollerine uygun işlemleri yapabilmelidirler. Örneğin müşteri aboneliğinin detayını sorgulayabilmeli, ödeme geçmişini görebilmeli, otoparka giriş çıkıl-ş kayıtlarını listeleyebilmeli. Aynı şekilde müşteri otopark abonelik talebini uygulama üzerinden kayıt açarak iletebilmelidir. Benzer şekilde muhasebe sorumlusu müşteri aboneliklerini gerçekleyip ödeme planını müşteriye iletebilmelidir. Otoparktan sorumlu müdür ise müşterileri takip edebilmeli, öneri ve şikayetleri takip edebilmeli, işletme sahibine aylık rapor sunmalıdır. İşletme sahibi ise müdür ve muhasebe sorumlusunun sahip olduğu tüm yetkilere sahip olmalıdır. Güvenlik görevlileri ise sadece araç ve müşteri bilgilerini sorgulayabilmeli, otopark alanının doluluk oranını takip edebilmeli, güvenlik ihlallerini kayıt altına alabilmelidir.
* Ödemeler peşin, kredi kartı veya banka havalesi ile yapılabilecektir. Yazılım tüm bu ödeme metotları için olanak sunmalıdır.
* Aboneler istedikleri zaman aboneliklerini iptal ettirebilirler. İptal edilen abonelik için geriye dönük geri iade yapılmayacaktır.
* Abonelik için şu bilgiler müşteriden alınmalıdır. Ad, Soyadı, TCK No, Araç Plaka no, Adres, Telefon, e-posta adresi ve Abonelik tercihi (aylık, yıllık). Bu Bilgilerden Ad, Soyadı, Araç Plaka No, Abonelik türü zorunlu ilken diğerleri opsiyoneldir.
* Ödemelerini geciktiren müşterilere otomatik sms ve e-posta gönderilecektir

Bu senaryo ve gereksinimler üzerinden aşağıdaki konuları içeren soruları cevaplayınız.

1. Tüm sistemin bir **kullanım durumu UML diyagramı** veya diyagramlarını (bölmek isterseniz) çiziniz. Bu diyagramda aktörler olarak müşteri, muhasebeci ve müdür olmalıdır. Diyagramınızda genişleme noktaları (extention point) örneği yer almasına dikkat ediniz.
2. Abonelik kaydı yaptırma sürecinin **aktivite UML diyagramı**’nı çiziniz.
3. Muhasebe sorumlusu ödeme raporu alacaktır, bu rapor alma sürecini **sekans UML diyagramı**

ile çiziniz.

1. Tüm sistemin, web ön yüzü olarak linux üzerinde “Apache Web Server”, arka plan sunucusu olarak linux üzerinde “Tomcat java uygulama sunucusu” ve veri tabanı olarak da Windows işletim sistemi üzerinde “SQL Server” kullanıldığını kabul ederek **dağıtım UML diyagramı**’nı çiziniz
2. Ana sınıflarınızı gösteren **sınıf UML diyagram**ını (birbirleri ile olan ilişkileri de göstererek) çiziniz. Sınıfların uygulama kodları hariç tanım kodlarını java dilinde yazınız.
3. Sınıflarınızı geliştirirken yaratımsal desenlerden **Singleton** ve **Builder** desenlerini kullandığınız örnek kodları yazınız. Bu kodlar denenmiş, çalışır olduğu görülmüş kodlar olmalıdır.
4. Müşteri için sms ve e-posta iletişim altyapısı için bir kütüphane kullanmanız gerekti. Ancak bu kütüphane çok karmaşık bu nedenle bir tasarım deseni ile basitleştirmek istiyorsunuz. Uygun yapıcı tasarım desenini kullanarak sendSMS ve sendEmail isimli iki public metod sunan yeni bir API sınıfı gerçekleyiniz. Kütüphane kodunun var olduğunu düşünün. Geliştireceğiniz metotları ise kod içeriğinde gerçek sms ve e-posta gönderme yerine system.output.println çıktısı ile yapıldığını düşünerek geliştirebilirsiniz.
5. Ödeme işlemini gerçekleyen kod parçanızı **decorator** yapıcı tasarım desenini kullanarak Java ile gerçekleyiniz. Kodların çalıştığı onaylanmalıdır. Deseni isterseniz daha uygun olacağını düşündüğünüz farklı bir sınıf için de uygulayabilirisiniz

# İp Uçları ve diğer konular:

1. Senaryoya uygun olduğunu düşündüğünüz eklemeler yapabilirsiniz.
2. Sistemde tek olarak ifade edilecek bir nesneyi singleton deseni için kullanabilirsiniz. (Örnek: kapalı otopark, işletme sahibi).
3. Müşterilerin oluşumunu builder deseni ile yapabilirsiniz. Dilerseniz başka sınıflarda uygun olabilir
4. UML çizimleri için istediğiniz ürünü kullanabilirsiniz. Diyagramları düzgün, süreci iyi ifade eden, bakıldığında anlaşılacak şekilde çiziniz.
5. Basit düşünün, sadece istenen kısımlara odaklanın, karmaşıklaştırmayın. 6- Ödev vize sınavı yerine geçecektir. Yıl sonuna katkısı %25 olacaktır.
6. Süresinde getirilmeyen ödev yapılmamış kabul edilecektir. Süre uzatımı olmayacaktır.
7. Ödev dijital çıktı ile Word dosyası olarak oluşturulacak ve sisteme eklenecektir. Yazdığınız kaynak kodlar da Word dosyası içinde yer almalıdır.
8. Ödev bireysel olarak hazırlanacaktır. Ekip veya grup çalışması olmayacaktır. Belli bir oranda kopya ödevler teslim edilmemiş sayılacaktır.
9. Yapayzeka çıktıları teste tabii olacak, böyle bir kullanımda ödev teslim edilmemiş sayılacaktır. 11- Son teslim tarihi: 17 Nisan 2023 saat: 18:00

# Değerlendirme Kriterleri:

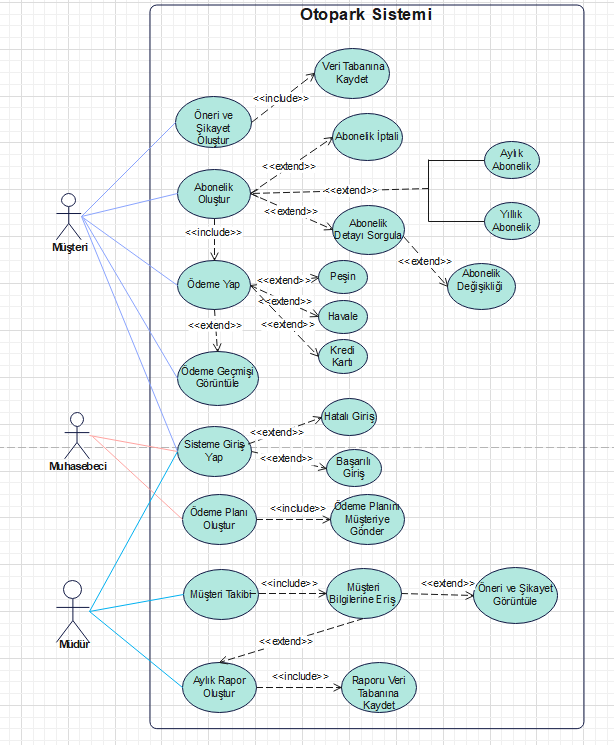
|  |  |  |  |
| --- | --- | --- | --- |
| Soru  No | Kriter | Ağırlık | Değerlendirme notu |
|  |  |  |  |
| 1 | Kullanım durumlarınını senaryodaki isterleri karşılayacak nitelikte olması. Kullanım durum  diyagramının anlaşılır ve ilişkileri gösterir şekilde tasarlanması | 10 |  |
| 2 | Aktivite diyagramının istenen süreç için eksiksiz çizilmesi, süreci tam ifade edecek tasarımın  yapılması | 10 |  |
| 3 | Sekans diyagramının istenen süreç için eksiksiz çizilmesi, süreci tam ifade edecek tasarımın  yapılması | 10 |  |
| 4 | Dağıtım diyagramının istenen yapı için eksiksiz çizilmesi, yapıyı tam ifade edecek tasarımın  yapılması | 10 |  |
| 5 | Temel ana sınıfların sınıf diyagramında ilişkileri de  doğru gösterecek şekilde çizilmesi | 15 |  |
| 6 | Singleton ve Builder desenlerinin kullanıldığı kod’un  çalışabilir ve desene uygun olması | 15 |  |
| 7 | Seçilen desenin amacına uygun şekilde  gerçeklenmesi | 15 |  |
| 8 | Decorator deseninin kullanıldığı kod’un çalışabilir ve  desene uygun olması | 15 |  |

Dr. Suat UĞURLU

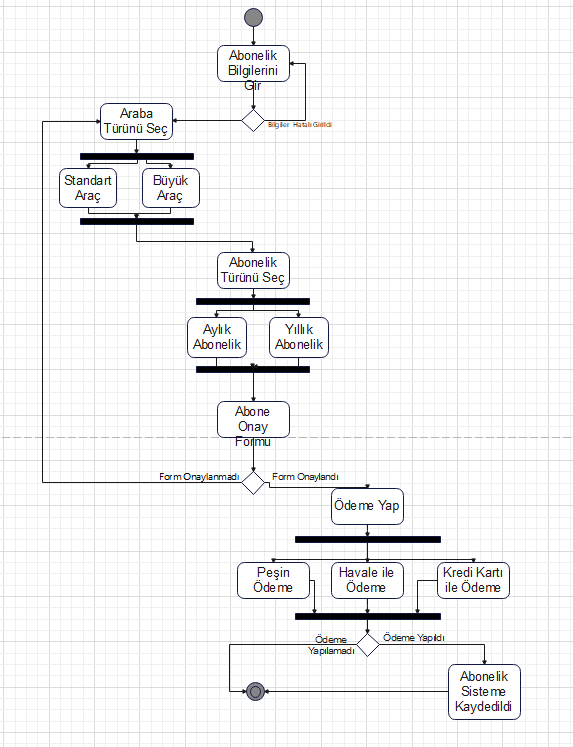
[Suatugurlu@topkapi.edu.tr](mailto:Suatugurlu@topkapi.edu.tr)

9 Nisan 2023

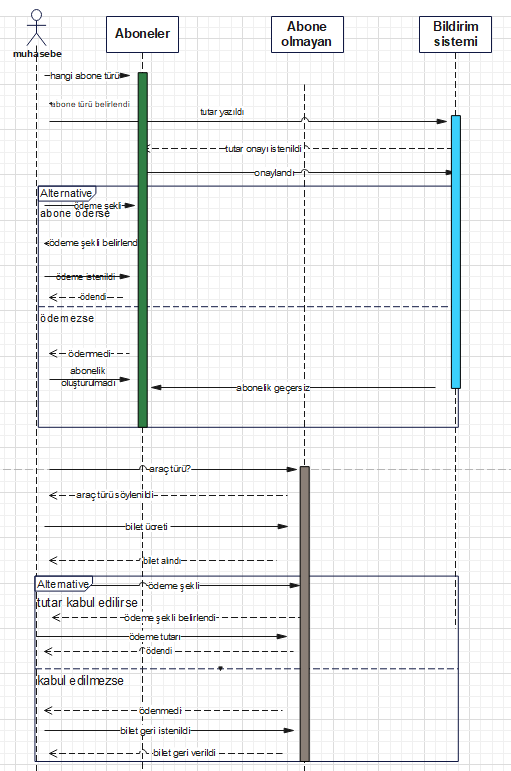
**1-) UseCase Diyagramı**

****

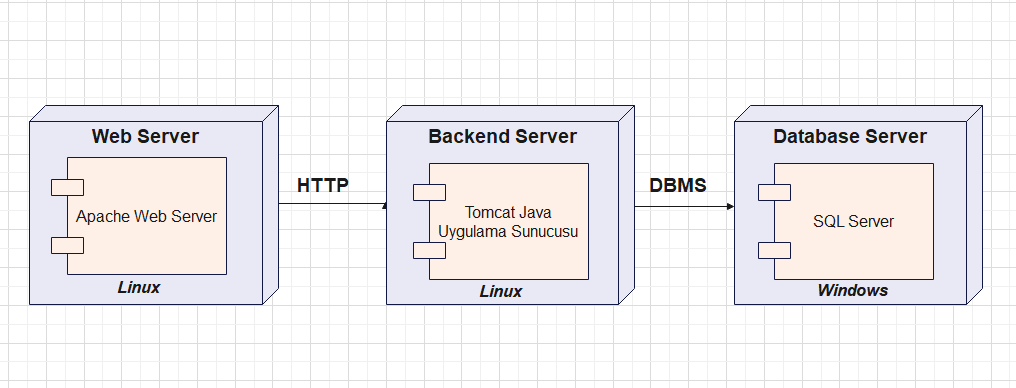
**1-)Aktivite Diyagramı**

****

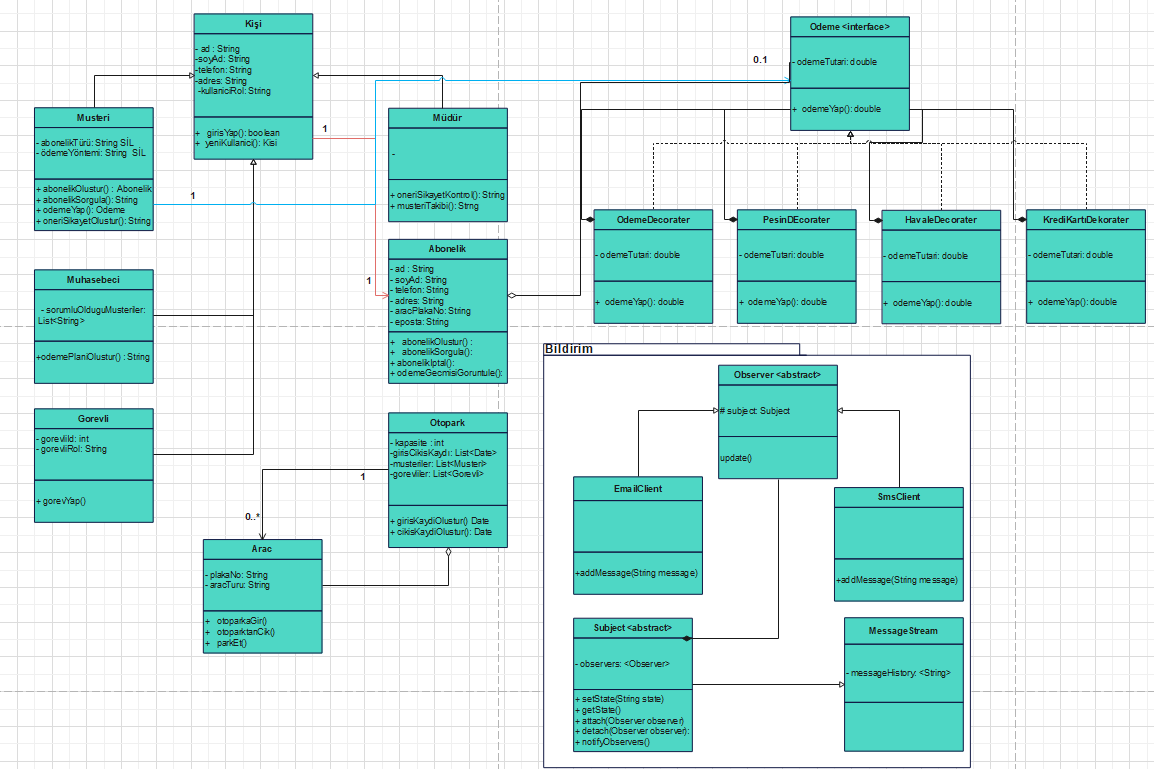
**3-) Sekans Diyagramı**

****

**4-) Dağıtım Diyagramı**

****

**5-) Sınıf Diyagramı**

****

**6.1-) Singleton Tasarım Deseni:**

*import* java.util.ArrayList;  
*import* java.util.Date;  
  
*// Otoparkın kendisinden 1 tane olabileceği için Singleton tasarım deseni kullanmak daha doğru olacktır.  
public class* Otopark {  
 *// Singleton otopark nesnesi  
 private static* Otopark *otopark* = *null*;  
  
 *// Otopark özellikleri  
 private int* kapasite,doluYerSayisi;  
 *private* ArrayList<Musteri> musteriler;  
 *private* ArrayList<Gorevli> gorevliler;  
 *private* ArrayList<Date> girisCikisKayitlari;  
  
 *private* Otopark(*int* kapasite) {  
 }  
  
 *// Singleton ınstancesını geriye döndüren metot  
 public static* Otopark getInstance(*int* kapasite) {  
 *if* (*otopark* == *null*) {  
 *otopark* = *new* Otopark(kapasite);  
 }  
 *return otopark*;  
 }  
  
 *public void* girisKaydiOlustur() {  
 System.out.println("Giris kaydi olusturuluyor...");  
 }  
  
 *public void* cikisKaydiOlustur() {  
 System.out.println("Cikis kaydi olusturuluyor...");  
 }  
  
  
 *public int* getKapasite() {  
 *return* kapasite;  
 }  
  
 *public void* setKapasite(*int* kapasite) {  
 *this*.kapasite = kapasite;  
 }  
  
 *public int* getDoluYerSayisi() {  
 *return* doluYerSayisi;  
 }  
  
 *public void* setDoluYerSayisi(*int* doluYerSayisi) {  
 *this*.doluYerSayisi = doluYerSayisi;  
 }  
  
 *public* ArrayList<Musteri> getMusteriler() {  
 *return* musteriler;  
 }  
  
 *public void* setMusteriler(ArrayList<Musteri> musteriler) {  
 *this*.musteriler = musteriler;  
 }  
  
 *public* ArrayList<Gorevli> getGorevliler() {  
 *return* gorevliler;  
 }  
  
 *public void* setGorevliler(ArrayList<Gorevli> gorevliler) {  
 *this*.gorevliler = gorevliler;  
 }  
  
 *public* ArrayList<Date> getGirisCikisKayitlari() {  
 *return* girisCikisKayitlari;  
 }  
  
 *public void* setGirisCikisKayitlari(ArrayList<Date> girisCikisKayitlari) {  
 *this*.girisCikisKayitlari = girisCikisKayitlari;  
 }  
}

**6.2-) Builder Tasarım Deseni:**

*//Abonelik sınıfında girilen bilgiler arasında opsiyonellik bulunduğu için Builder tasarım deseni kullanmak daha işlevsel olacaktır. Musteri sınıfının içerisinde kullanılıyor.  
public class* Abonelik {  
 *private* String ad,soyad,tckNo,aracPlakaNo,adres,telefon,eposta;  
 *private* Enums.AbonelikTipi abonelikTuru;  
  
 *private* Abonelik(Builder builder) {  
 *this*.ad = builder.ad;  
 *this*.soyad = builder.soyad;  
 *this*.tckNo = builder.tckNo;  
 *this*.aracPlakaNo = builder.aracPlakaNo;  
 *this*.adres = builder.adres;  
 *this*.telefon = builder.telefon;  
 *this*.eposta = builder.eposta;  
 *this*.abonelikTuru = builder.abonelikTuru;  
 }  
  
 *public* String getAd() {  
 *return* ad;  
 }  
  
 *public* String getSoyad() {  
 *return* soyad;  
 }  
  
 *public* String getTckNo() {  
 *return* tckNo;  
 }  
  
 *public* String getAracPlakaNo() {  
 *return* aracPlakaNo;  
 }  
  
 *public* String getAdres() {  
 *return* adres;  
 }  
  
 *public* String getTelefon() {  
 *return* telefon;  
 }  
  
 *public* String getEposta() {  
 *return* eposta;  
 }  
  
 *public* Enums.AbonelikTipi getAbonelikTuru() {  
 *return* abonelikTuru;  
 }  
  
 *//inner Class  
 public static class* Builder {  
 *private* String ad,soyad,tckNo,aracPlakaNo,adres,telefon,eposta;  
 *private* Enums.AbonelikTipi abonelikTuru;  
  
 *public* Builder(String ad, String soyad, String aracPlakaNo, Enums.AbonelikTipi abonelikTuru) {  
 *this*.ad = ad;  
 *this*.soyad = soyad;  
 *this*.aracPlakaNo = aracPlakaNo;  
 *this*.abonelikTuru = abonelikTuru;  
 }  
  
 *public* Builder setTckNo(String tckNo) {  
 *this*.tckNo = tckNo;  
 *return this*;  
 }  
 *public* Builder setAdres(String adres) {  
 *this*.adres = adres;  
 *return this*;  
 }  
 *public* Builder setTelefon(String telefon) {  
 *this*.telefon = telefon;  
 *return this*;  
 }  
 *public* Builder setEposta(String eposta) {  
 *this*.eposta = eposta;  
 *return this*;  
 }  
 *public* Abonelik build() {  
 *return new* Abonelik(*this*);  
 }  
 }  
}

**7-) Bildirim Gönderme Sınıfı:**

* **Bildirim gönderme sınıfında Observer tasarım desenini kullandım.**
* **Observer Ana Sınıfı**

*package* Bildirim;  
  
*public abstract class* Observer {  
  
 *protected* Subject subject;  
  
 *abstract void* update();  
  
}

* **Subject Sınıfı**

*package* Bildirim;  
  
*import* java.util.ArrayList;  
*import* java.util.*List*;  
  
*public abstract class* Subject {  
  
 *private List*<Observer> observers = *new* ArrayList<>();  
  
 *abstract void* setState(String state);  
 *abstract* String getState();  
  
 *public void* attach(Observer observer) {  
 observers.add(observer);  
 }  
  
 *public void* detach(Observer observer) {  
 observers.remove(observer);  
 }  
  
 *public void* notifyObservers() {  
 *for* (Observer observer : observers) {  
 observer.update();  
 }  
 }

* **EmailClient Sınıfı**

*package* Bildirim;  
  
*public class* EmailClient *extends* Observer {  
  
 *public* EmailClient (Subject subject) {  
 *this*.subject = subject;  
 subject.attach(*this*);  
 }  
  
 *public void* addMessage(String message) {  
 subject.setState(message + " - email gönderildi");  
 }  
  
 @Override  
 *void* update() {  
 System.out.println("Email Stream: " + subject.getState());  
 }  
}

* **SmsClient Sınıfı**
* *package* Bildirim;  
    
  *public class* SmsClient *extends* Observer {  
    
   *public* SmsClient (Subject subject) {  
   *this*.subject = subject;  
   subject.attach(*this*);  
   }  
    
   *public void* addMessage(String message) {  
   subject.setState(message + " sms gönderildi");  
   }  
    
   @Override  
   *void* update() {  
   System.out.println("Sms Stream: " + subject.getState());  
   }  
  }
* **MessageStream Sınıfı**

*package* Bildirim;  
  
*import* java.util.ArrayDeque;  
*import* java.util.Deque;  
  
*public class* MessageStream *extends* Subject {  
  
 *private* Deque<String> messageHistory = *new* ArrayDeque<>();  
  
 @Override  
 *void* setState(String message) {  
 messageHistory.add(message);  
 *this*.notifyObservers();  
 }  
  
 @Override  
 String getState() {  
 *return* messageHistory.getLast();  
 }  
}

* **BildirimGonder Sınıfı => Bu sınıf Observer desenini uygulayan kod oluyor. Bu Sınıfın nesnesini de Main sınıfında çağırarak işlemleri gerçekleştirdim.**

*package* Bildirim;  
  
*public class* BildirimGonder {  
*public void* gecikmeBildirimi()  
{  
 Subject subject = *new* MessageStream();  
 SmsClient smsClient = *new* SmsClient(subject);  
 EmailClient emailClient = *new* EmailClient(subject);  
  
 subject.setState("Bildirim Gönderildi");  
  
 smsClient.addMessage("Ödemeniz Gecikti. Lütfen ödemenizi yapın!");  
 emailClient.addMessage("Ödemeniz Gecikti. Lütfen ödemenizi yapın!");  
  
}  
  
}

* **Decorator Tasarım Deseni**

Odeme işlemlerinde birden fazla seçenek olduğu için Decorator desenini kullandım**.**

* **Odeme Interface’i**
* *// Odeme ınterface'i  
  interface Odeme* {  
   *void* odemeYap();  
  }
* **OdemeDecorater Sınıfı**

*// Ana Decorator soyut sınıfımız  
abstract class* OdemeDecorator *implements Odeme* {  
 *private Odeme* decoratedOdeme;  
  
 *public* OdemeDecorator(*Odeme* decoratedOdeme) {  
 *this*.decoratedOdeme = decoratedOdeme;  
 }  
  
 *public* OdemeDecorator() {  
  
 }  
  
 @Override  
 *public void* odemeYap() {  
 decoratedOdeme.odemeYap();  
 }  
}

* **HavaleDecorator**

*class* HavaleDecorator *extends* OdemeDecorator {  
  
 *public* HavaleDecorator(*Odeme* decoratedOdeme) {  
 *super*(decoratedOdeme);  
 }  
  
 @Override  
 *public void* odemeYap() {  
 *super*.odemeYap();  
 havaleYap();  
 }  
  
 *private void* havaleYap() {  
 System.out.println("Havale yapılıyor...");  
 }  
}

* **BankaHavalesi sınıfı**
* *class* BankaHavalesi *implements Odeme* {  
   @Override  
   *public void* odemeYap() {  
   System.out.println("Banka Havalesi ile ödeme yapıldı.");  
   }  
  }
* **KrediKartiDecorator**

*//Kredi kartı ile ödeme yapmak için Decorator sınıfımız  
class* KrediKartiDecorator *extends* OdemeDecorator {  
 *public* KrediKartiDecorator(*Odeme* decoratedOdeme) {  
 *super*(decoratedOdeme);  
 }  
  
 @Override  
 *public void* odemeYap() {  
 *super*.odemeYap();  
 System.out.println("Kredi Kartı ile ödeme yapıldı.");  
 }  
  
}

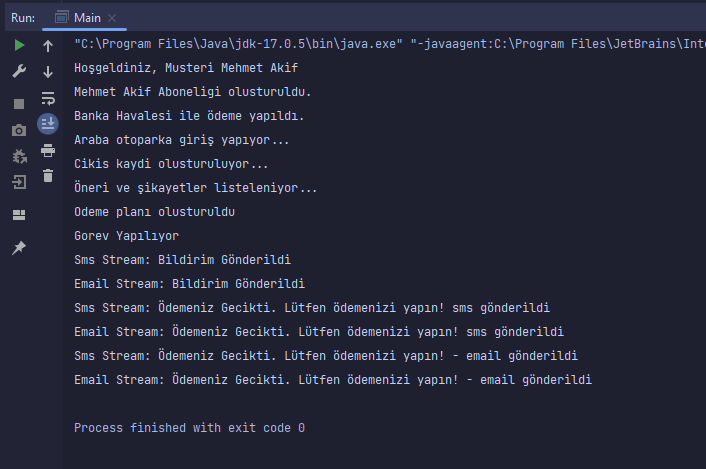
* **PesinDecorator**

*// //Peşin ödeme yapmak için Decorator sınıfımız  
class* PesinDecorator *extends* OdemeDecorator {  
 *public* PesinDecorator(*Odeme* decoratedOdeme) {  
 *super*(decoratedOdeme);  
 }  
  
 @Override  
 *public void* odemeYap() {  
 *super*.odemeYap();  
 System.out.println("Peşin ödeme yapıldı.");  
 }  
}

* **Main fonksiyonu => Tüm sınıfların çalıştığını test etmek için her sınıftan nesne türetip denedim.**

*import* Bildirim.BildirimGonder;  
  
*public class* Main {  
 *public static void* main(String[] args) {  
  
 Musteri musteri = *new* Musteri("Mehmet","Akif","Musteri");  
 musteri.Login(musteri);  
 musteri.abonelikOlustur();  
 musteri.odemeYap();  
  
 Arac a1 = *new* Arac();  
 a1.otoparkaGir();  
  
 Otopark o1= Otopark.*getInstance*(150);  
 o1.cikisKaydiOlustur();  
  
 Mudur m1 = *new* Mudur();  
 m1.oneriSikyetTakibi();  
  
 Muhasebeci muhasebeci1 = *new* Muhasebeci();  
 muhasebeci1.odemePlaniOlustur();  
  
 Gorevli gorevli1 = *new* Gorevli();  
 gorevli1.gorevYap();  
  
 BildirimGonder bildirim = *new* BildirimGonder();  
 bildirim.gecikmeBildirimi();  
  
 }  
}

**Ekran Çıktısı**

****