Yazılım Tasarımı ve Mimarisi Final

Tasarım Desenleri

Genel olarak tasarım desenleri:

- Creational Patterns (Oluşturucu Desenler)
    1. Singleton
    2. Factory
    3. Abstract Factory
    4. Builder
- Structural Patterns (Yapısal Desenler)
    1. Facade
    2. Adapter
    3. Bridge
    4. Composite
- Behavioral Patterns (Davranışsal Desenler)
    1. Observer
    2. Chain of Responsibility

3. Visitor
4. Strategy
5. State

Oluşturucu (Creational) Desenler

1. Singleton

- Bir sınıftan tek bir örnek oluşturulmak istendiğinde kullanılır.
- Sistem çalıştığı sürece örnekten ikinci bir örnek oluşmaz.
- Database erişim işleri gibi tekrar nesne oluşturulmaması gerektiğinde kullanılır.

1. Yöntem → Multithread kullanılmıyorsa

```
public class Singleton {
  private static Singleton nesne = new Singleton();

  private Singleton() {
  ...
  }

  public static Singleton getInstance() {
    if(nesne == null)
      nesne = new Singleton();
    return nesne;
  }
}
```

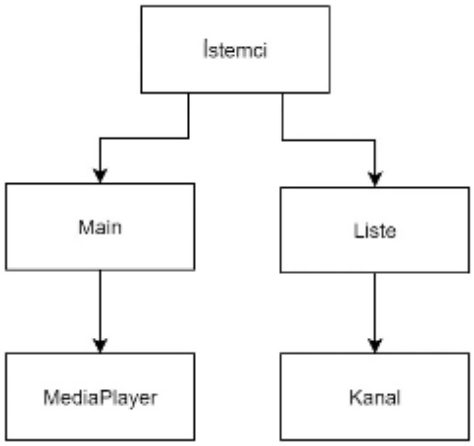
```
// main'den çağrılışı:
singleton = Singleton.getInstance();
```

2. Yöntem → Multithread kullanılıyorsa

```
public class Singleton {
  private static Singleton nesne;
  private static Object kanalkontrol = new Object;

  public Singleton() {
```

```
  ...
  }

  public static Singleton getInstance() {
    if(nesne == null) {
      lock(kanalkontrol) {
        if(nesne == null) {
          nesne = new Singleton();
        }
      }
    }
    return nesne;
  }
}
```

Singleton Uygulaması




```
public interface Activity {
  public void onCreate();
}

public class MainActivity implements Activity {
  private MediaPlayer MediaPlayer;

  @Override
  public void onCreate() {
    mediaPlayer = MediaPlayer.getInstance();
  }

  public void oynat(Kanal kanal) {
    mediaPlayer.kanalSec(kanal);
```

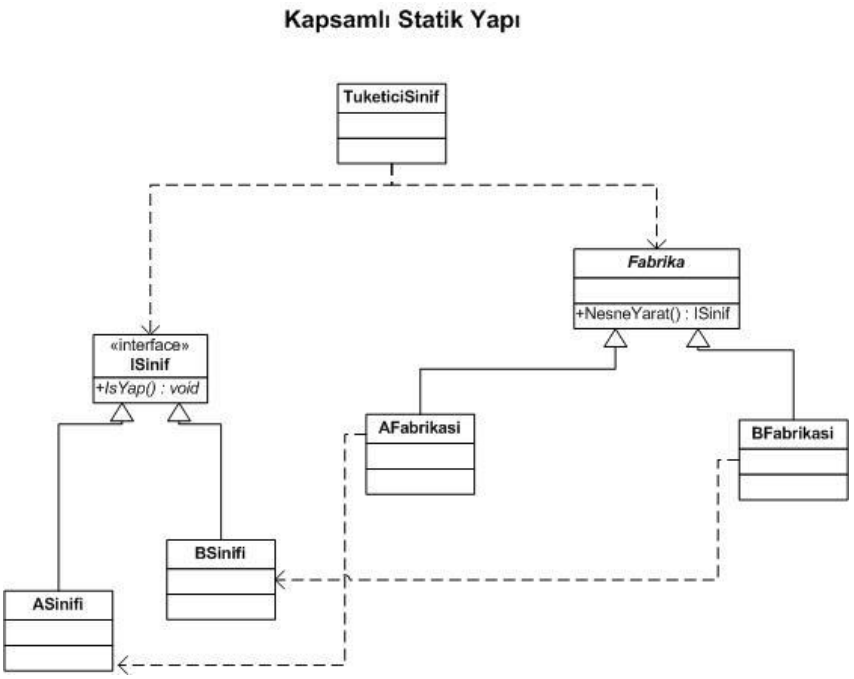
```
    mediaPlayer.oynat();
  }

  public void durdur() {
    mediaPlayer.durdur();
  }
}

public class ListeActivity {
  ArrayList<Kanal> kanalListesi;

  @Override
  public void onCreate() {
    kanalListesi = kanalListesi();
  }

  public ArrayList<Kanal> kanalListesi() {
    ArrayList<Kanal> kanalListesi = new ArrayList<>();
    Kanal kanal1 = new Kanal("Kanal 1", "www.kanal1.com");
    Kanal kanal2 = new Kanal("Kanal 2", "www.kanal2.com");

    kanalListesi.add(kanal1);
    kanalListesi.add(kanal2);
    return kanalListesi;
  }

  public Kanal kanalSec(int index) {
    return kanalListesi.get(index);
  }
}

public class Kanal {
  private String isim, link;

  public Kanal(String isim, String link) {
    setIsim(isim);
    setLink(link);
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim){
    this.isim = isim;
  }

  public String getLink() {
    return isim;
  }

  public void setLink(String link){
    this.link = link;
  }
}
```

```
// sürekli çalışmasını sağlar -> Runnable
public class MediaPlayer implements Runnable {
  private Kanal kanal;
  private boolean kontrol;
  // Singleton 1. adım
  private static MediaPlayer mediaPlayer;

  // Singleton 2. adım
  private MediaPlayer() { }

  // Singleton 3. adım
  public static MediaPlater getInstance() {
      if(mediaPlayer == null) {
        mediaPlayer = new MediaPlayer();
      }
      return mediaPlayer;
  }

  public void kanalSec(Kanal kanal) {
    this.kanal = kanal;
  }

  public void oynat() {
    kontrol = true;
    new Thread(this).start();
  }

  public void durdur() {
    kontrol = false;
  }

  @Override
  public void run() {
    while(kontrol) {
      Main.Sleep(0200);
      System.out.println(kanal.getIsim() + " kanali " + kanal.getLink() + " linki ");
    }
  }
}

public class Main {

  public static void main(String[] args) {
    MainActivity mainActivity = new MainActivity();
    ListeActivity listeActivity = new ListeActivity();

    mainActivity.onCreate();
    listeActivity.onCreate();

    Kanal kanal = listeActivity.kanalSec(0);
    mainActivity.onCreate();
    mainActivity.oynat(kanal);

    Main.sleep(5000);
```

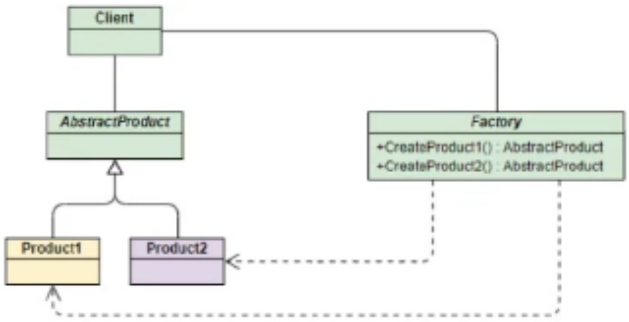
```
    listeActivity.onCreate();
    kanal2 = listeActivity.kanalSec(1);
    mainActivity.onCreate();
    mainActivity.durdur();
    mainActivity.oynat(kanal2);
  }

  public static void Sleep(int sure) {
    try {
      Thread.sleep(sure);
    } catch(InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

2. Factory

- Client → Factory → Class
- Factory sınıfının temel görevi client'ın istediği nesneyi hangi sınıftan nasıl türeteceğini ondan soyutlamaktır.
- Nesneler temel sınıflar üzerinden değil factory üzerinden türetilir.
- Genel UML diyagramı:



Factory Uygulaması

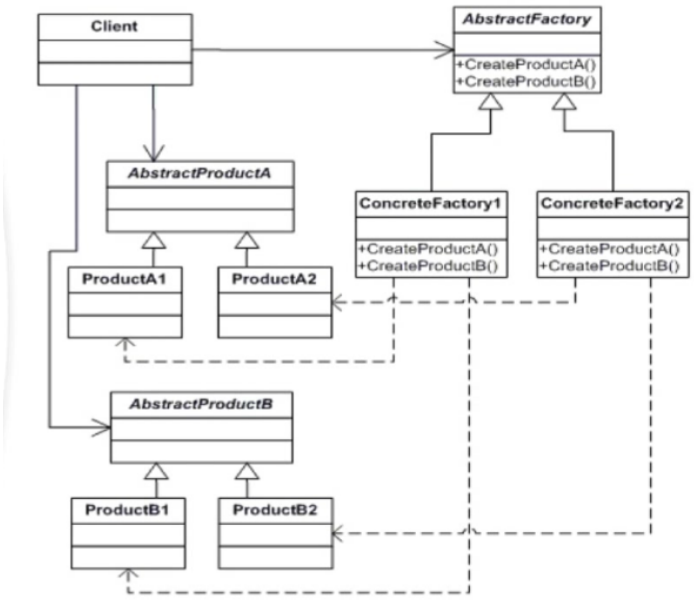



```
namespace fabrikaOrnegiKumas {
  interface IFabrika {
    IKumas fabrikaMetodu(string urun);
  }

  class KumasFabrika : IFabrika {
    public IKumas fabrikaMetodu(string urun) {
      switch(urun) {
        case "keten": return new KetenKumas();
        case "kadife": return new KadifeKumas();
        default: return new KadifeKumas();
      }
    }
  }

  interface IKumas {
    string kumasUret();
  }

  class KetenKumas : IKumas {
    public string kumasUret() {
      return "keten kumas";
    }
  }

  class KadifeKumas : IKumas {
    public string kumasUret() {
      return "kadife kumas";
    }
  }

  class Program {
    static void Main(string[] args) {
      string uretilenKumas;
```

```
        IFabrika kumasFabrika = new KumasFabrika();
        Ikumas kumas = kumasFabrika.fabrikaMetodu("keten");
        uretilenKumas = kumas.kumasUret();
        Console.WriteLine(uretilenKumas + " uretildi.");

        kumas = kumasFabrika.fabrikaMetodu("kadife");
        uretilenKumas = kumas.kumasUret();
        Console.WriteLine(uretilenKumas + " uretildi.");

        Console.ReadKey();
    }
  }
}
```

3. Abstract Factory

- Birbirine bağlı olan nesnelerin oluşturulması en etkin şekilde çözmeyi hedefler.
- Dolap ve masa → Mobilya Grubu
- Kazak ve gömlek → Giysi Grubu
- Genel UML:



Abstract Factory Uygulaması

```
namespace SoyutFabrikaOrnegiKanepe {
  interface ISoyutUrunKumas {
    string kumasUret();
  }

  class KadifeKumas : ISoyutUrunKumas {
    public string kumasUret() {
      return "kadife kumas";
    }
  }

  class KetenKumas : ISoyutUrunKumas {
    public string kumasUret() {
      return "keten kumas";
    }
  }

  interface ISoyutUrunIskelet {
    string iskeletUret();
  }

  class MetalIskelet : ISoyutUrunIskelet {
    public string iskeletUret() {
      return "metal iskelet";
    }
  }

  class AhsapIskelet : ISoyutUrunIskelet {
    public string iskeletUret() {
      return "ahsap iskelet";
    }
  }

  // fabrika sınıfından farkı burada
  interface ISoyutFabrika {
    ISoyutUrunKumas kumasGetir();
    ISoyutUrunIskelet iskeletGetir();
  }

  class SomutFabrika1 : ISoyutFabrika {
    public ISoyutUrunKumas kumasGetir() {
      return new KetenKumas();
    }

    public ISoyutUrunIskelet iskeletGetir() {
      return new AhsapIskelet();
    }
  }

  class AraIstemci {
    private ISoyutUrunIskelet iskelet;
    private ISoyutUrunKumas kumas;
```

```
    public AraIstemci(ISoyutFabrika soyutFabrika) {
      kumas = soyutFabrika.kumasGetir();
      iskelet = soyutFabrika.iskeletGetir();
    }

    public void calistir() {
      string iskeleletUret = iskelet.iskeletUret();
      string kumasUret = kumas.kumasUret();

      Console.WriteLine("Koltuk " + iskeletUret + " ve " + kumasUret + " uretilmistir.");
    }
  }

  class Program {
    static void Main(string[] args){
      ISoyutFabrika soyutFabrika = new SomutFabrika1();
      AraIstemci araIstemci = new AraIstemci(soyutFabrika);
      araIstemci.calistir();

      Console.ReadKey();
    }
  }
}
```

4. Builder

- Varlıkların birçok alt parçanın birleşiminden oluştuğunu varsayar.
- Bu pattern; aynı kompleks ürünün farklı parçalarla oluşturulup farklı sunumlarının elde edilebilmesini sağlar.

Builder Uygulaması

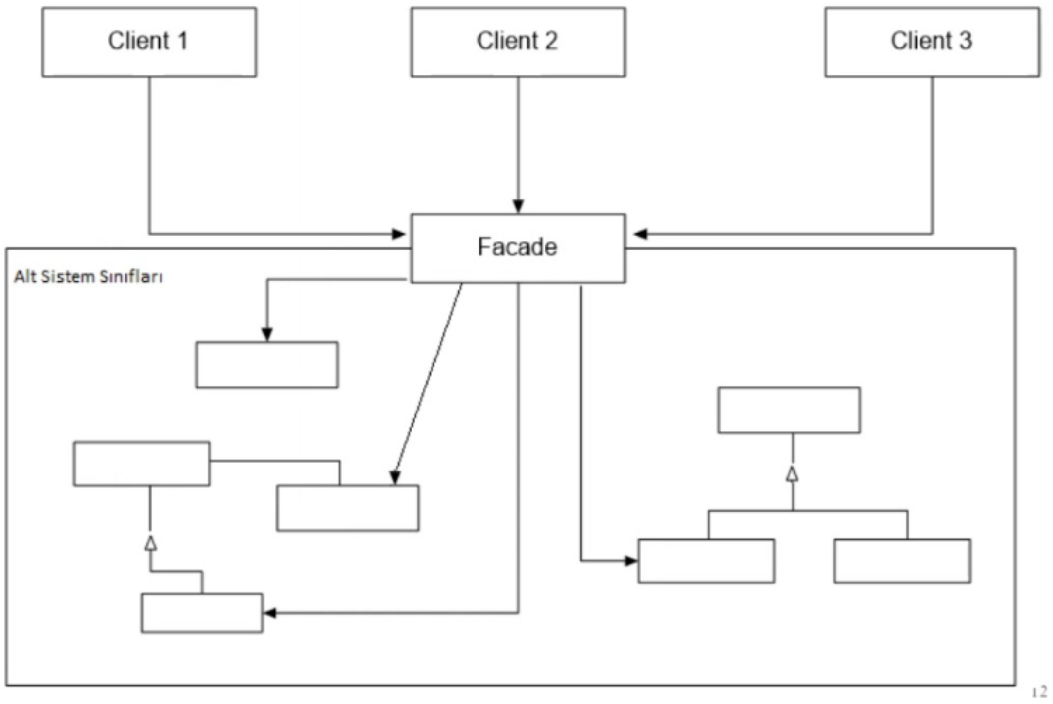
```
public class BankAccount {
  public static class Builder {
    private long accountNumber;
    private String owner;
    private String branch;
    private double balance;
    private double interestRate;

    public Builder(long accountNumber) {
      this.accountNumber = accountNumber;
    }

    public Builder withOwner(String owner) {
      this.owner = owner;
      return this;
    }
```

```
    public Builder atBranch(String branch) {
      this.branch = branch;
      return this;
    }

    public Builder openingBalance(double balance) {
      this.balance = balance;
      return this;
    }

    public Builder atRate(double interestRate) {
      this.interestRate = interestRate;
      return this;
    }

    public BankAccount build() {
      BankAccount account = new BankAccount();
      account.accountNumber = this.accountNumber;
      account.owner = this.owner;
      account.branch = this.branch;
      account.balance = this.balance;
      account.interestRate = this.interestRate;

      return account;
    }
  }

  private BankAccount() {
    // ...
  }

  // Getters and setters
}

public class Main {

  public static void main(String[] args) {
    BankAccount account = new BankAccount.Builder(1234L);
    account.withOwner("Marge");
    account.atBranch("Springfield");
    account.openingBalance(100);
    account.atRate(2.5);
    account.build();
  }
}
```

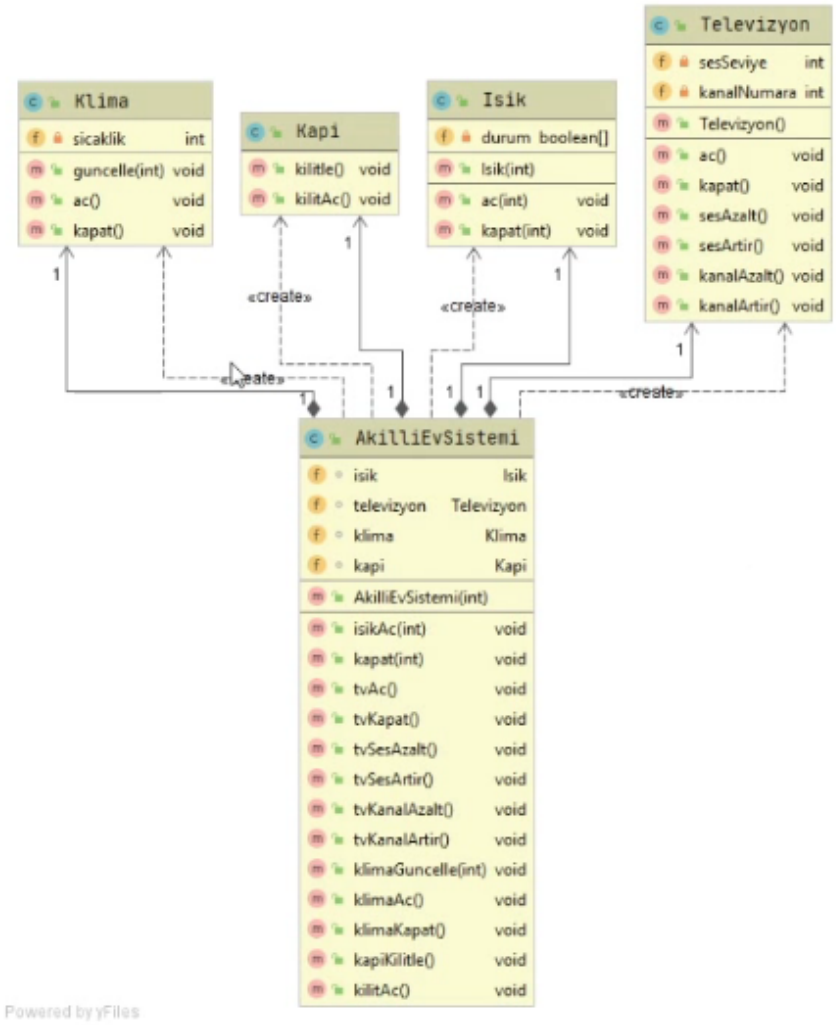
Yapısal (Structural) Desenler

1. Facade

- Var olan nesneye yeni bir yüz (cephe) katma anlamında kullanılır.
- Çok katmanlı mimariye sahip uygulamaların geliştirilmesinde kullanılır. Böylece katmanlar birbirinden bağımsız olarak geliştirilebilir.
- Alt sistemler direkt olarak kullanılmaz. Alt sistemlerin kullanılmasını sağlayan arayüzler geliştirilir.
- Genel UML:



Facade Uygulaması




```java
public class Klima {
    private int sicaklik;

    public void guncelle(int sicaklik) {
      this.sicaklik = sicaklik;
      System.out.println("Klima " + sicaklik + " dereceye ayarlandi.");
    }

    public void ac() {
      System.out.println("Klima acildi");
    }

    public void kapat() {
      System.out.println("Klima kapandi");
    }
}
```

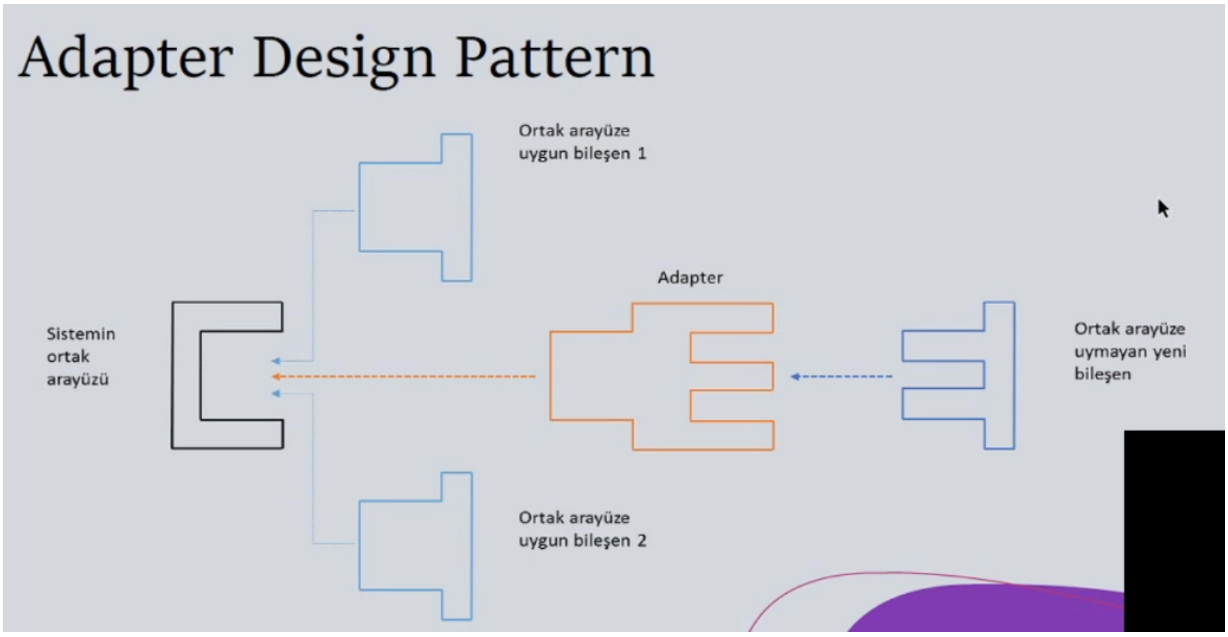
```java
public class Kapi {
  public void kilitle() {
    System.out.println("Kapi kilitlendi");
  }

  public void kilidiAc() {
    System.out.println("Kapi kilidi acildi");
  }
}

public class Isik {
  private boolean durum[];

  public Isik(int adet) {
    durum = new boolean[adet];
  }

  public void ac(int numara) {
    durum[numara] = true;
    System.out.println(numara + " numarali isik acildi.");
  }

  public void kapat(int numara) {
    durum[numara] = false;
    System.out.println(numara + " numarali isik kapandi.");
  }
}

public class Televizyon {
  private int sesSeviye, kanalNumara;

  public Televizyon(){
    sesSeviye = 10;
    kanalNumara = 1;
  }

  public void ac() {
    System.out.println("Televizyon acildi");
  }

  public void kapat() {
    System.out.println("Televizyon kapatildi");
  }

  public void sesAzalt() {
    sesSeviye -= 1;
    System.out.println("Televizyon ses seviyesi azaltildi." + sesSeviye);
  }

  public void sesArttir() {
    sesSeviye += 1;
    System.out.println("Televizyon ses seviyesi arttirildi." + sesSeviye);
  }

  public void kanalAzalt() {
```

```
    kanalNumara -= 1;
    System.out.println("Televizyon kanal numarasi azaltildi." + kanalNumara);
  }

  public void kanalArttir() {
    kanalNumara += 1;
    System.out.println("Televizyon kanal numarasi arttirildi." + kanalNumara);
  }
}

// Facade burada
public class AkilliEvSistemi {
  Klima klima;
  Kapi kapi;
  Isik isik;
  Televizyon televizyon;

  public AkilliEvSistemi() {
    klima = new Klima();
    kapi = new Kapi();
    isik = new Isik();
     televizyon = new Televizyon();
  }

  public void klimaGuncelle(int sicaklik) {
    klima.guncelle(sicaklik);
  }

  public void klimaAc() {
    klima.ac();
  }

  public void klimaKapat() {
    klima.kapat();
  }

  public void kapiKilitle() {
    kapi.kilitle();
  }

  public void kapiKilidiAc() {
    kapi.kilidiAc();
  }

  public void isikAc(int numara) {
    isik.ac(numara);
  }

  public void isikKapat(int numara) {
    isik.kapat(numara);
  }

  public void televizyonAc() {
    televizyon.ac();
  }
```

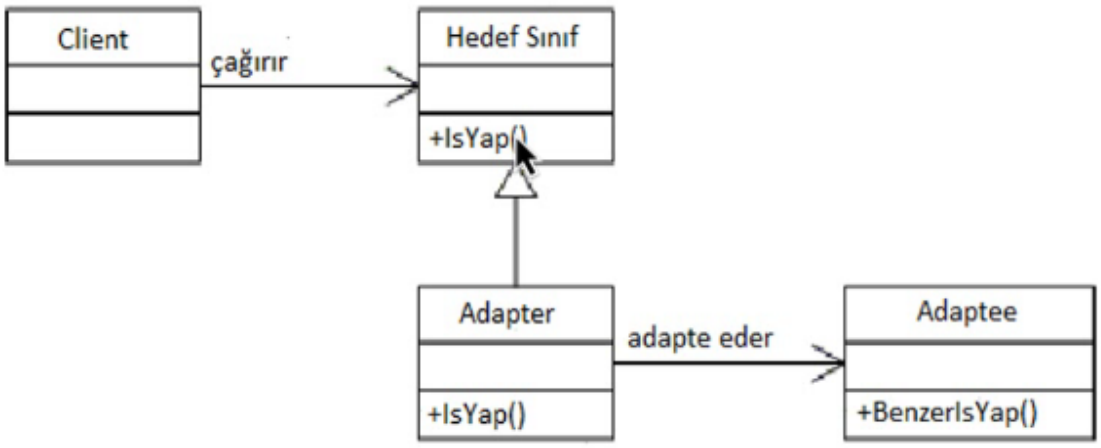
```
  public void televizyonKapat() {
    televizyon.kapat();
  }

  public void televizyonSesAzalt() {
    televizyon.sesAzalt();
  }

  public void televizyonSesArttir() {
    televizyon.sesArttir();
  }

  public void televizyonKanalAzalt() {
    televizyon.kanalAzalt();
  }

  public void televizyonKanalArttir() {
    televizyon.kanalArttir();
  }

  public void sinemaModu() {
    televizyonAc();
    televizyonSesArttir();
    for(int i = 0; i < 5; i++) {
      isikKapat(i);
    }
    klimaKapat();
  }
}

public class Main {
  public static void main(String[] args) {

    AkilliEvSistemi akilliEvSistemi = new AkilliEvSistemi(5);
    akilliEvSistemi.klimaGuncelle(25);
    akilliEvSistemi.kapiKilitle();
    akilliEvSistemi.isikAc();
    akilliEvSistemi.sinemaModu();
  }
}
```

2. Adapter

- Birbiriyle çalışmayan desenlerin birbiriyle çalışmasını sağlar.
- Bu işlemi ise bir sınıfın interface'ini diğer bir interface'e dönüştürerek yapar.



- Genel UML:



Adapter Uygulaması

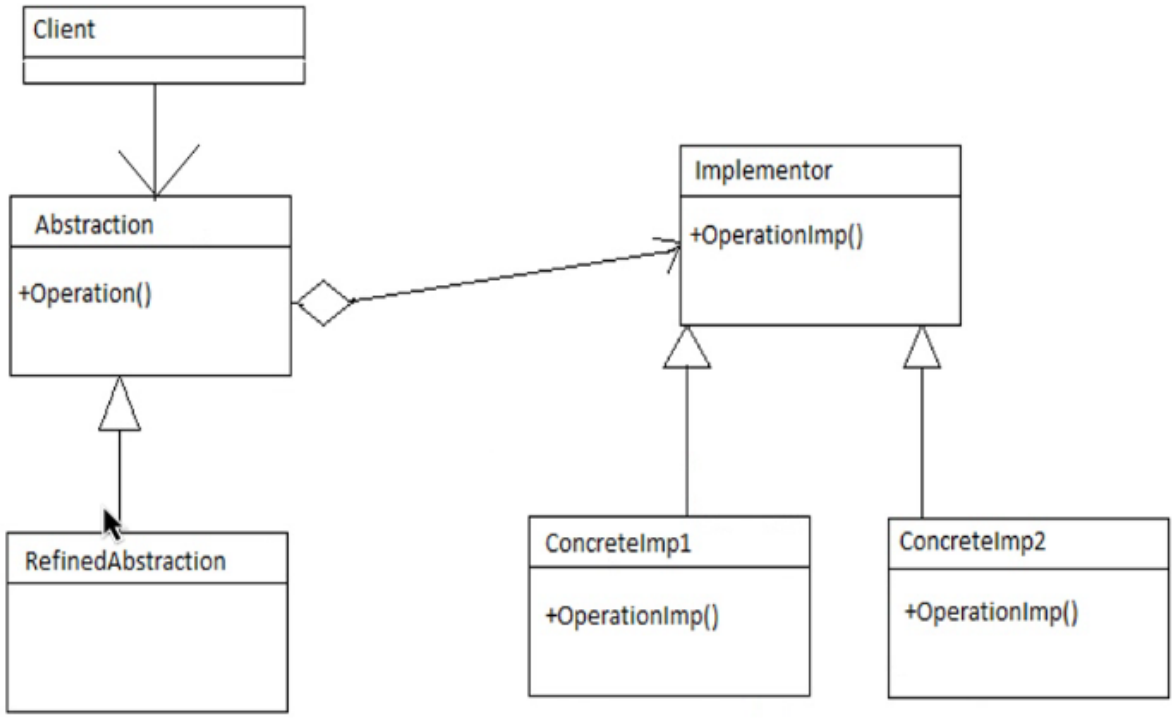
```
namespace AdapterDeseniOrnegi {
  interface IYedekleyici {
    void Kaydet(string kaynakKlasorKonumu, string hedefKlasorKonumu);
  }

  class DiskYedekleyici : IYedekleyici {
    public void Kaydet(string kaynakKlasorKonumu, string hedefKlasorKonumu) {
```

```
      Console.WriteLine(kaynakKlasorKonumu + " konumundaki dosyalar diskte " +
      hedefKlasorKonumu + " icine yedeklendi.");
    }
  }

  class FlashYedekleyici : IYedekleyici {
    public void Kaydet(string kaynakKlasorKonumu, string hedefKlasorKonumu) {
      Console.WriteLine(kaynakKlasorKonumu + " konumundaki dosyalar flashta " +
      hedefKlasorKonumu + " icine yedeklendi.");
    }
  }

  class UzakHedefeKayit {
    public void UzakHedefeKaydet(string kaynakKlasorKonumu, string hedefKlasorKonumu) {
      BaglantiKur(hedefKlasorKonumu);
      DosyalariGonder(kaynakKlasorKonumu);
      BaglantiyiKapat();
    }

    private void BaglantiKur(string hedefKlasorKonumu){
      Console.WriteLine(hedefKlasorKonumu + " ile baglanti kuruldu.");
    }

    private void DosyalariGonder(string kaynak) {
      Console.WriteLine(kaynak + " klasorundeki dosyalar hedef konuma gonderildi");
    }

    private void BaglantiyiKapat() {
      Console.WriteLine("Baglantiniz sonlandirildi");
    }
  }

  // Adapter deseni burada
  class UzakHedefeKayitAdapter : IYedekleyici {
    private UzakHedefeKayit uzakHedefeKayit;

    public void Kaydet(string kaynakKlasorKonumu, string hedefKlasorKonumu) {
      uzakHedefeKayit = new UzakHedefeKayit();
      uzakHedefeKayit.UzakHedefeKaydet(kaynakKlasorKonumu, hedefKlasorKonumu);
    }
  }

  class Program {
    static void Main(string[] args) {

      string kaynak = "C:\\YedeklenecekKlasor";
      IYedekleyici yedekleme;
      yedekleme = new DiskYedekleyici();
      yedekleme.Kaydet(kaynak, "D:\\HedefDiskKonumu");

      yedekleme = new FlashYedekleyici();
      yedekleme.Kaydet(kaynak, "U:\\HedefDiskKonumu");

      yedekleme = new UzakHedefeKayitAdapter();
      yedekleme.Kaydet(kaynak, "www.google.com\\hedefKonum");
```

```
            Console.ReadKey();
        }
    }
}
```

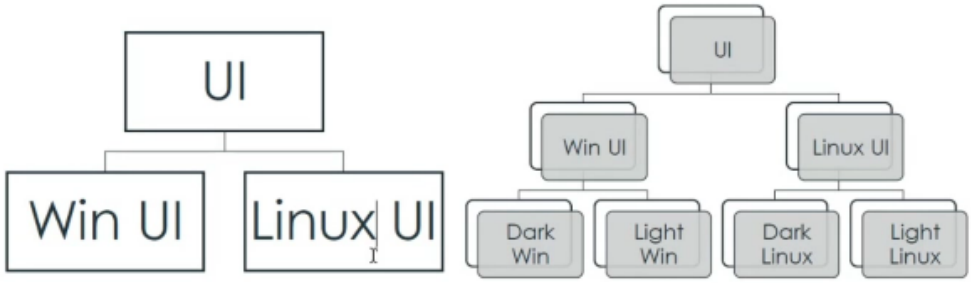
3. Bridge

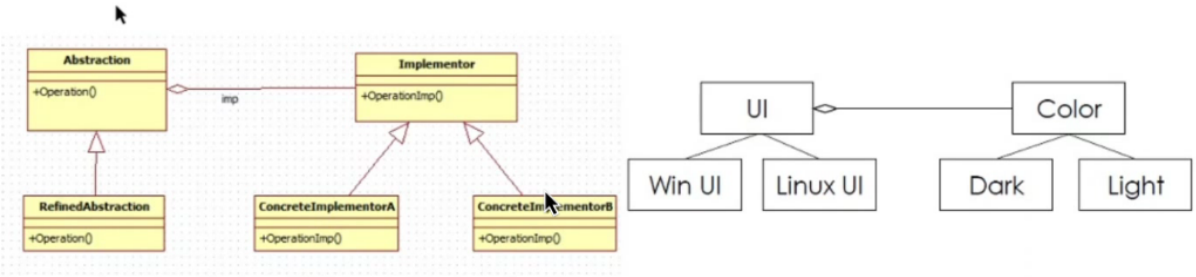
- Soyutlanılan nesneler ile işi gerçekleyecek somut nesneler arasında köprü kurar.
- Soyut sınıflar ve işi yapacak sınıflatı birbirinden ayırdığı için iki sınıf tipinde yapılacak bir değişiklik birbirini etkilemez.
- Hiyerarşi dikey yönde değil, yatay yönde büyüyecektir.
- Örnek: Rapor hazırlayan ve bu raporları farklı formatlarda sisteme kaydeden bir işlev.
- Genel UML:



Bridge Uygulaması

Problem-Örnek



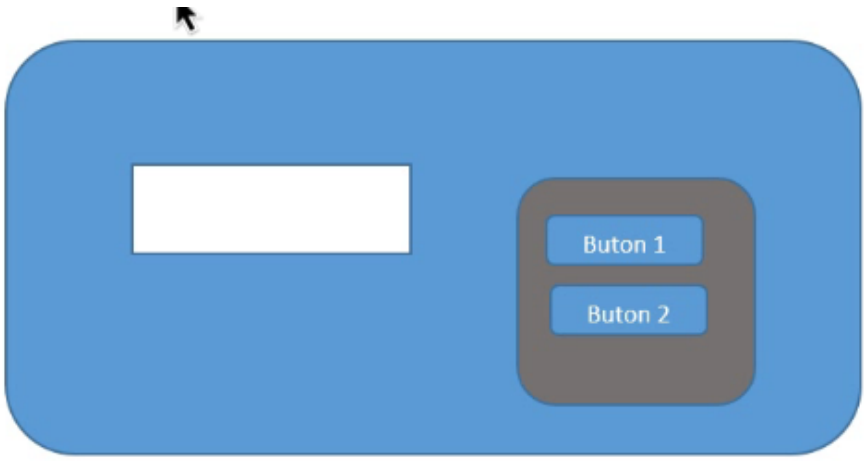



```
namespace BridgeOrnek {
  interface IRenk {
    string renkOlustur();
  }

  class KoyuRenk : IRenk {
    public string renkOlustur() {
      return "koyu renk";
    }
  }

  class AcikRenk : IRenk {
    public string renkOlustur() {
      return "acik renk";
    }
  }

  abstract class UI {
    public abstract void uiOlustur();
  }
```

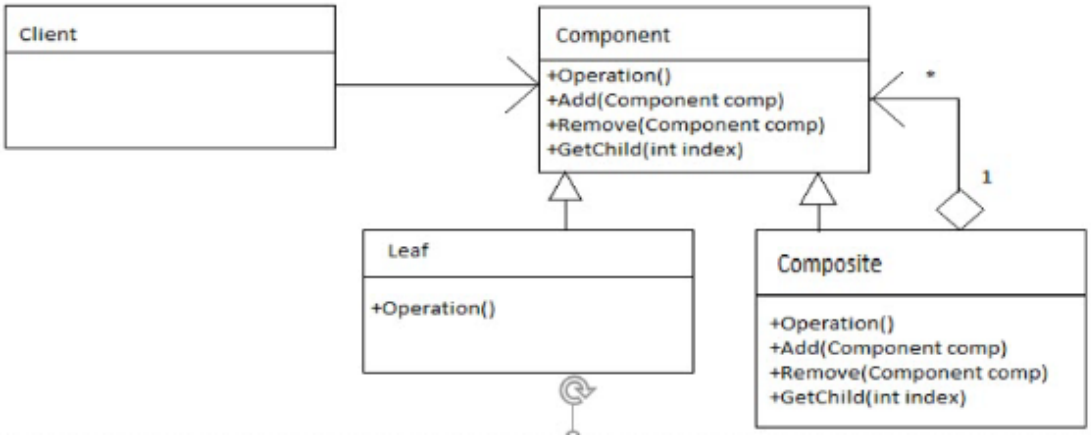
```
  class WinUI : UI {
    IRenk renk;

    public WinUI(IRenk renk) {
      this.renk = renk;
    }

    public override void uiOlustur() {
      string renkParametresi = renk.renkOlustur();
      Console.WriteLine(renkParametresi + " ile WinUI olusuturuldu.");
    }
  }

  class LinuxUI : UI {
    IRenk renk;

    public LinuxUI(IRenk renk) {
      this.renk = renk;
    }

    public override void uiOlustur() {
      string renkParametresi = renk.renkOlustur();
      Console.WriteLine(renkParametresi + " ile LinuxUI olusuturuldu.");
    }
  }

  class Program {
    static void Main(string[] args) {
      IRenk renk = new KoyuRenk();
      UI windowsUI = new WinUI(renk);
      windowsUI.uiOlustur();

      Console.ReadKey();
    }
  }
}
```

4. Composite

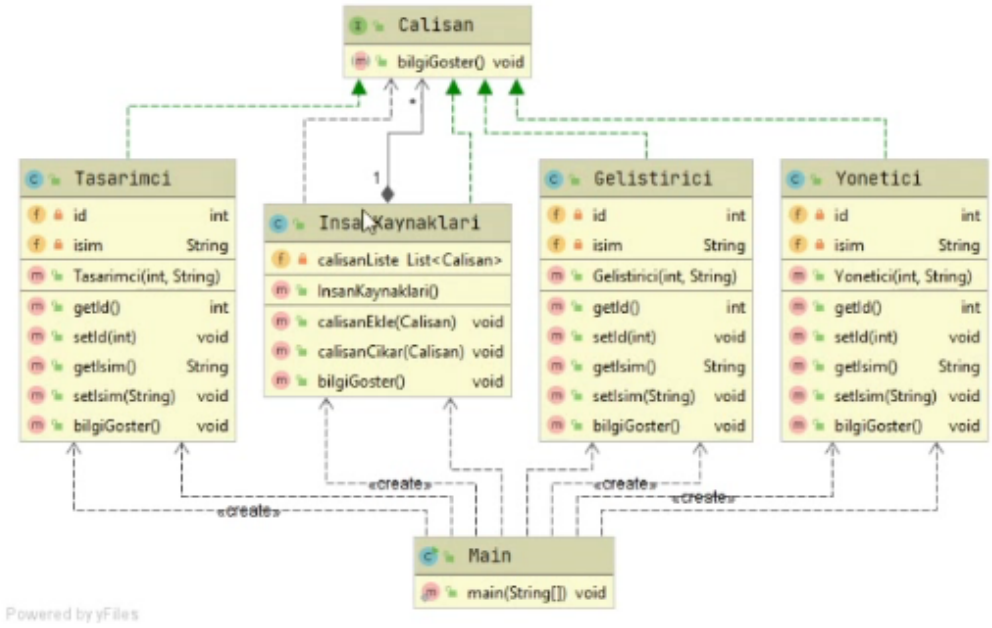
- Dinamik bir yapı oluşturulur.
- İç içe bir yapı vardır.
- Nesneler ağaç yapısına göre düzenlenir.
- Sistemin bütünü parçalardan oluşur.
- Türleri birbirinden farklı nesneler bir araya getirilir.



- Genel UML:



Composite Uygulaması




```java
public interface Calisan {
  void bilgiGoster();
}

public class Tasarimci implements Calisan {
  private int kurumId;
  private int isim;
  private String tasarimciBilgi;

  public Tasarimci(int kurumId, String isim) {
    setKurumId(kurumId);
    setIsim(isim);
  }

  public int getKurumId() {
    return kurumId;
  }

  public void setKurumId(int kurumId) {
    this.kurumId = kurumId;
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim) {
```

```
    this.isim = isim;
  }

  public String getTasarimciBilgi() {
    return tasarimciBilgi;
  }

  public void setTasarimciBilgi(String tasarimciBilgi) {
    this.tasarimciBilgi = tasarimciBilgi;
  }

  public void tasarimYap() {
    System.out.println(getIsim() + " - " + getTasarimciBilgi());
  }

  @Override
  public void bilgiGoster() {
    System.out.println(getKurumId() + " - " + getIsim());
  }
}

public class Gelistirici implements Calisan {
  private int kurumId;
  private String isim;
  private String gelistiriciBilgi;

  public Gelistirici(int kurumId, String isim) {
    setKurumId(kurumId);
    setIsim(isim);
  }

  public int getKurumId() {
    return kurumId;
  }

  public void setKurumId(int kurumId) {
    this.kurumId = kurumId;
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim) {
    this.isim = isim;
  }

  public String getGelistiriciBilgi() {
    return gelistiriciBilgi;
  }

  public void setTasarimciBilgi(String gelistiriciBilgi) {
    this.gelistiriciBilgi = gelistiriciBilgi;
  }
```

```
  public void gelistirmeYap() {
    System.out.println(getIsim() + " - " + getTasarimciBilgi());
  }

  @Override
  public void bilgiGoster() {
    System.out.println(getKurumId() + " - " + getIsim());
  }
}

public class Yonetici implements Calisan {
  private int kurumId;
  private String isim;
  private String yoneticiBilgi;

  public Yonetici(int kurumId, String isim) {
    setKurumId(kurumId);
    setIsim(isim);
  }

  public int getKurumId() {
    return kurumId;
  }

  public void setKurumId(int kurumId) {
    this.kurumId = kurumId;
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim) {
    this.isim = isim;
  }

  public String getYoneticiBilgi() {
    return yoneticiBilgi;
  }

  public void setYoneticiBilgi(String yoneticiBilgi) {
    this.yoneticiBilgi = yoneticiBilgi;
  }

  public void yoneticilikYap() {
    System.out.println(getIsim() + " - " + getGelistiriciBilgi());
  }

  @Override
  public void bilgiGoster() {
    System.out.println(getKurumId() + " - " + getIsim());
  }
}

public class InsanKaynaklari implements Calisan {
```

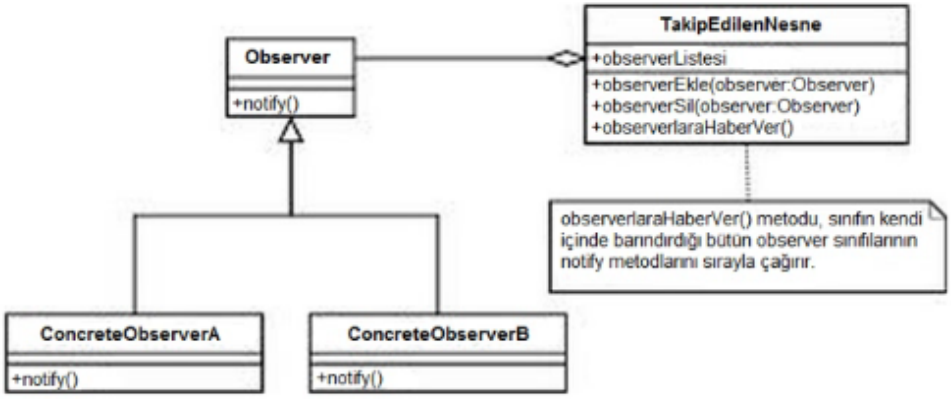
```
  private List<Calisan> calisanListe;

  public InsanKaynaklari() {
    calisanListe = new ArrayList<>();
  }

  public void calisanEkle(Calisan calisan) {
    calisanListe.add(calisan);
  }

  public void calisanCikar(Calisan calisan) {
    calisanListe.remove(calisan);
  }

  @Override
  public void bilgiGoster() {
    for(Calisan calisan : calisanListe) {
      calisan.bilgiGoster();
    }
  }
}

public class Main {
  public static void main(String[] args) {

    Gelistirici g1 = new Gelistirici(202, "Gelistiric ornek2");
    InsanKaynaklari gelistiriciler = new InsanKaynaklari();
    gelistiriciler.calisanEkle(g1);

    InsanKaynaklari calisanlar = new InsanKaynaklari();
    calisanlar.calisanEkle(gelistiriciler);

    calisanlar.bilgiGoster();
  }
}
```

Davranışsal Desenler

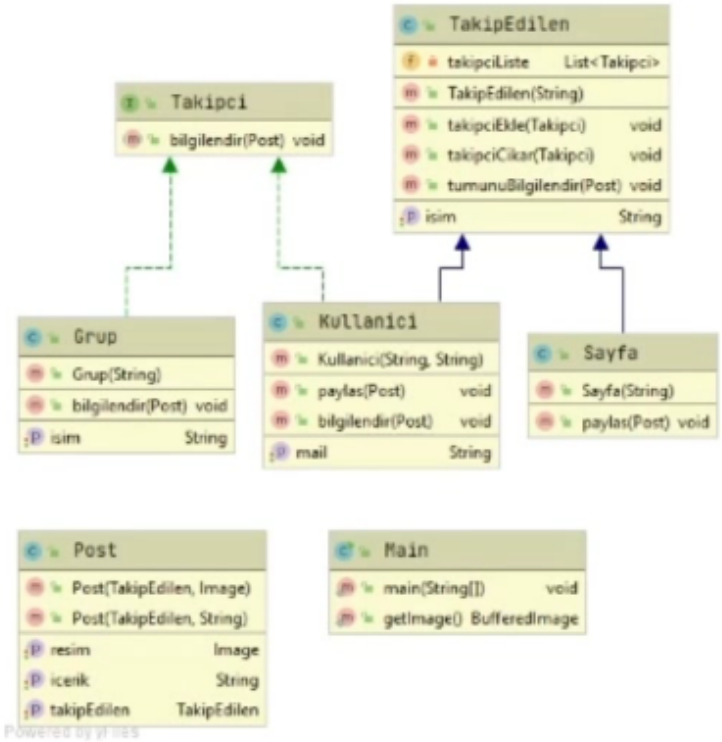
1. Observer

- Bir nesnede meydana gelen değişikliklerde, içinde bulundurduğu nesnelere haber gönderir.
- En sık kullanılan tasarım desenidir.
- Gözlemci sınıf → Haberdar olması gereken sınıftır.
- Hepsi aynı base sınıftan türerler ve takip edilmek istenen sınıfın içerisinde bu base sınıf tipinden bir liste bylunur.

- Seçilen sınıf değiştirildiğinde diğer nesnelere haber verilir ve kendilerini değiştirirler.
- Genel UML:



Observer Uygulaması



```
public class Kullanici extends Paylasimci implements Takipci {
  private String mail;

  public Kullanici(String isim, String mail) {
    super(isim);
    setMail(mail);
  }

  public String getMail() {
    return mail;
  }

  public void setMail(String mail) {
    this.mail = mail;
  }

  @Override
  public void bilgilendir(Post post) {
    System.out.println(getIsim() + " kullanicisina " + post.getPaylasimci().getIsim()
    + " paylasim mesaj olarak gonderildi");
  }

  @Override
  public void paylas(Post post) {
    System.out.println(getIsim() + " kullanicisi " + post.getBaslik() + " paylasti.");
    tumunuBilgilendir(post);
  }
}

public class Post {
  private String baslik, icerik;
  private Image resim;
  private Paylasimci paylasimci;

  public Post(String baslik, String icerik, Paylasimci paylasimci) {
    setBaslik(baslik);
    setIcerik(icerik);
    setPaylasimci(paylasimci);
  }

  public Post(String baslik, Image resim, Paylasimci paylasimci) {
    setBaslik(baslik);
    setResim(resim);
    setPaylasimci(paylasimci);
  }

  public String getBaslik() {
    return baslik;
  }

  public void setBaslik(String baslik) {
    this.baslik = baslik;
  }
```

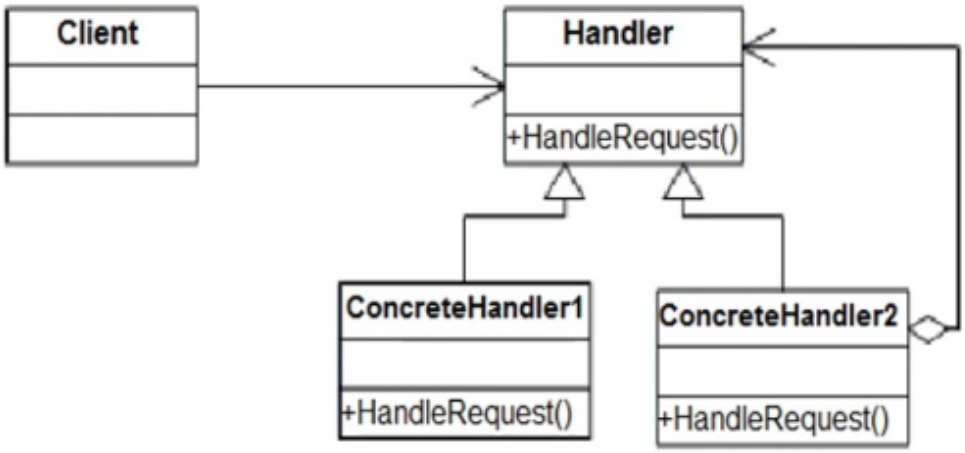
```
  public String getIcerik() {
    return mail;
  }

  public void setIcerik(String icerik) {
    this.icerik = icerik;
  }

  public Image getResim() {
    return image;
  }

  public void setResim(Image resim) {
    this.resim = resim;
  }

  public Paylasimci getPaylasimci() {
    return paylasimci;
  }

  public void setPaylasimci(Paylasimci paylasimci) {
    this.paylasimci = paylasimci;
  }
}

public abstract class Paylasimci {
  private String isim;
  private List<Takipci> takipciListe;

  public Paylasimci() {
    setIsim(isim);
    takipciListe = new ArrayList<>();
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim) {
    this.isim = isim;
  }

  public abstract void paylas(Post post);

  public void takipciEkle(Takipci takipci) {
    takipciListe.add(takipci);
  }

  public void takipciCikar(Takipci takipci) {
    takipciListe.remove(takipci);
  }

  public void tumunuBilgilendir(Post post) {
    for(Takipci takipci : takipciListe) {
```

```
      takipci.bilgilendir(post);
    }
  }
}

public class Sayfa extends Paylasimci {
  public Sayfa(String isim) {
    super(isim);
  }

  @Override
  public void paylas(Post post) {
    System.out.println(getIsim() + " sayfasi " + post.getBaslik() + " paylasti.");
    tumunuBilgilendir(post);
  }
}

public class Grup implements Takipci{
  private String isim;

  public Grup(String isim) {
    setIsim(isim);
  }

  public String getIsim() {
    return isim;
  }

  public void setIsim(String isim) {
    this.isim = isim;
  }

  @Override
  public void bilgilendir(Post post) {
    System.out.println(getIsim() + " grubuna " + post.getPaylasimci().getIsim()
    + " paylasim mail olarak gonderildi");
  }
}

// Observer bilgilendirme
public interface Takipci {
  void bilgilendir(Post post);
}

public class Main {

  public static void main(String[] args) {
    Kullanici kullanici1 = new Kullanici("Betul", "betul@mail.com");
    Kullanici kullanici2 = new Kullanici("Alper", "alper@mail.com");
    Grup grup = new Grup("Yazilim");
    Sayfa sayfa = new Sayfa("Teknoloji");
    Post post = new Post("Resim", getImage(), kullanici1);

    kullanici1.takipciEkle(kullanici);
    kullanici1.takipciEkle(grup);
```

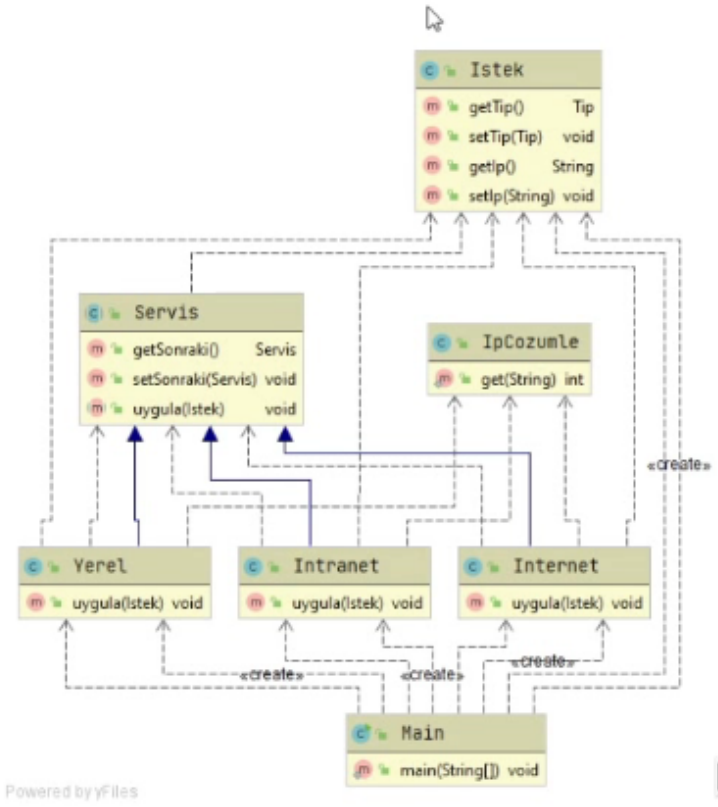
```
    sayfa.takipciEkle(kullanici1);
    sayfa.takipciEkle(kullanici2);

    kullanici1.paylas(post);

    sayfa.paylas(new Post("baslik", "icerik", sayfa));
  }

  public static BufferedImage getImage() {
    String imagePath = "Test Image";
    try {
      return ImageIo.read(new File(imagePath));
    } catch(IOException e){
      return null;
    }
  }
}
```

2. Chain of Responsibility

- Bu desen, istemcinin bir işi hakkında bilgi sahibi olmadığı bir dizi nesneye talep göndererek yaptırmasını şematize eder.
- Nesneler ile istemci nesneleri arasında doğrudan ilişki olmamalıdır.
- Örneğin: video oynatan farklı video oynatıcıları.
- Sorumluluğu üstlenecek olan nesneler (handler nesneler) belirli bir sırayla ve duruma göre görevi bir sonraki nesneye iletmesinden gelir.
- İstemcinin bilgisi dışında zincire yeni nesneler eklenebilir.
- Genel UML:



Chain of Responsibility Uygulaması



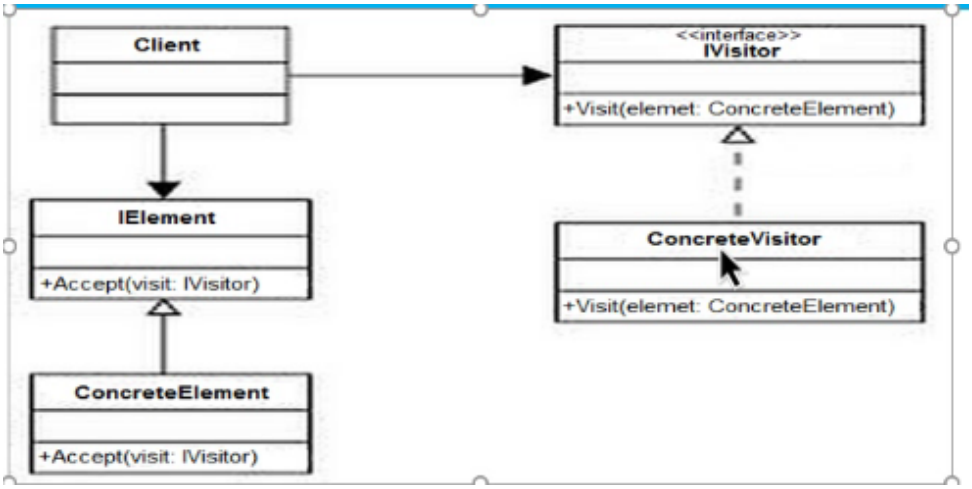

```
public abstract class Servis {
  public void uygula(Istek istek);
```

```
  // Chain of responsibility burada
  private Servis sonraki;

  public Servis getSonraki() {
    return sonraki;
  }

  public Servis setSonraki() {
    this.sonraki = sonraki;
  }
}

public class IpCozumle {
  public static int get(String ip) {
    String[] parcalar = ip.split("\\.");
    if(parcalar.length > 0) {
      return Integer.parseInt(parcalar[0]);
    }
    return 300;
  }
}

public class Istek {
  enum Tip {
    GET, POST, PUT, DELETE
  }

  private Tip tip;
  private String ip;

  public Istek(String ip, Tip tip) {
    setIp(ip);
    setTip(tip);
  }

  public Tip getTip() {
    return tip;
  }

  public void setTip(Tip tip) {
    this.tip = tip;
  }

  public String getIp() {
    return ip;
  }

  public void setIp(String ip) {
    this.ip = ip;
  }
}

public class Yerel extends Servis {
  @Override
```

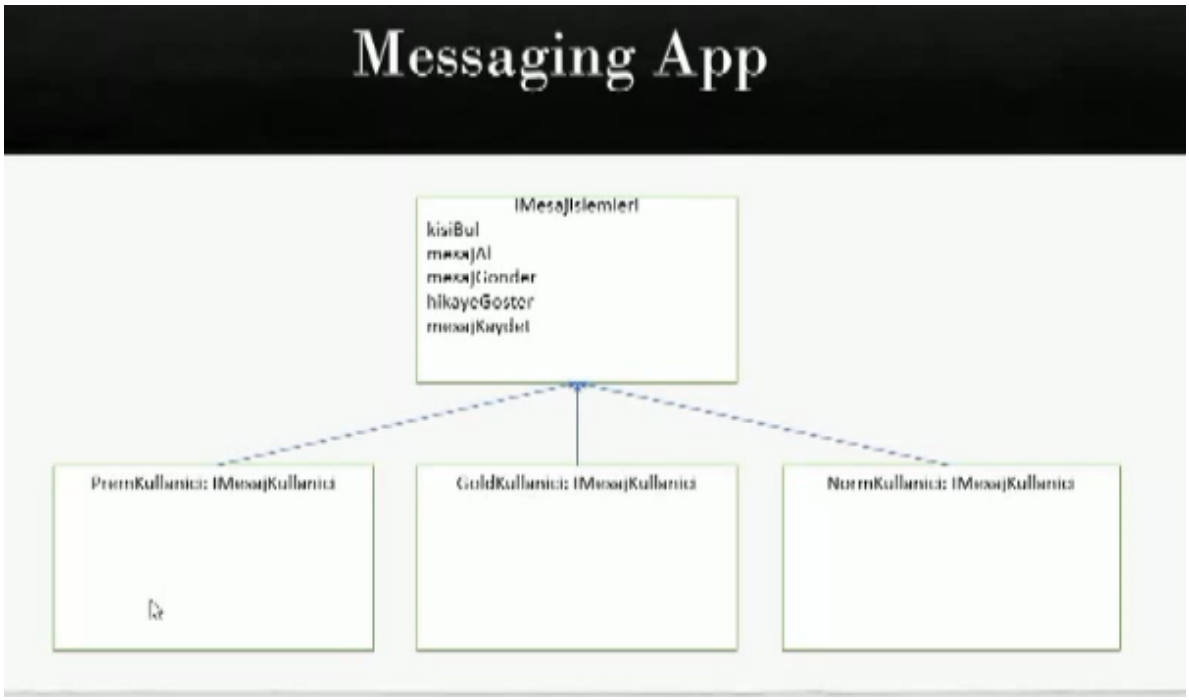
```
  public void uygula(Istek istek) {
    if(IpCozumle.get(istek.getIp()) < 100) {
      System.out.println("Yerel agdaki " + istek.getIp() + " adresine " +
      istek.getTip() + " istegi yapildi.");
    } else {
      if(getSonraki() != null) {
        getSonraki().uygula(istek);
      }
    }
  }
}

public class Intranet extends Servis {
  @Override
  public void uygula(Istek istek) {
    int ilkIp = IpCozumle.get(istek.getIp());
    if(ilkIp < 200 && ilkIp > 100) {
      System.out.println("Intranet agdaki " + istek.getIp() + " adresine " +
      istek.getTip() + " istegi yapildi.");
    } else {
      if(getSonraki() != null) {
        getSonraki().uygula(istek);
      }
    }
  }
}

public class Internet extends Servis {
  @Override
  public void uygula(Istek istek) {
    int ilkIp = IpCozumle.get(istek.getIp());
    if(ilkIp < 255 && ilkIp > 200) {
      System.out.println("Internet agindaki " + istek.getIp() + " adresine " +
      istek.getTip() + " istegi yapildi.");
    } else {
      if(getSonraki() != null) {
        getSonraki().uygula(istek);
      }
    }
  }
}

public class Main {
  public static void main(String[] args) {
    Yerel yerel = new Yerel();
    Intranet intranet = new Intranet();
    Internet internet = new Internet();

    yerel.setSonraki(intranet);
    intranet.setSonraki(internet);

    yerel.uygula(new Istek("50.0.0.1", Istek.Tip.GET));
  }
}
```

3. Visitor

- Bir uygulamada kurulu da olabilir, yapıyı kurarken de olabilir.
- Mevcut yapının içerisine kurulurken esnek bir yapının oluşturulması için uygulanan bir çözümdür.
- Bir sınıfa başka bir davranış eklemek istediğimizde kullanılır.
- Genel UML:



Visitor Uygulaması



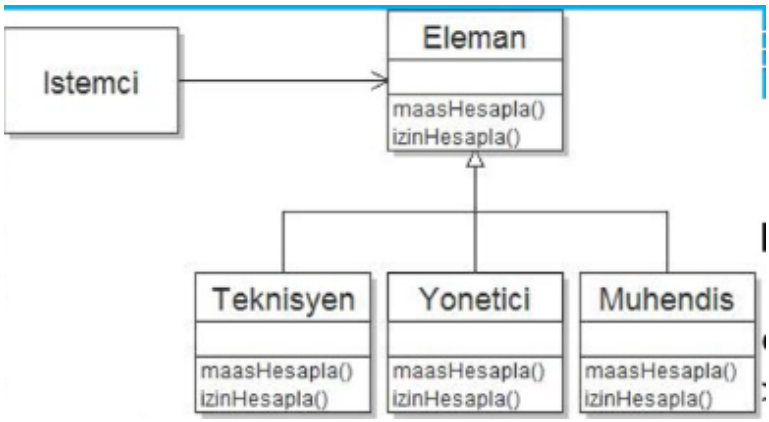

```
namespace visitor_ornegi {
  // visitable
  interface IKullaniciMesajIslem {
    void kisiBul(string kisi);
    void mesajAl();
    void mesajGonder(string message);
    // accept visitor
    void accept(Visitor visitor);
  }

  class NormalKullanici : IKullaniciMesajIslem {
    public void kisiBul(string kisi) {
      Console.WriteLine("normal kullanici icin " + kisi + " adli kisi bulundu");
    }

    public void mesajAl() {
      Console.WriteLine("normal kullanici icin mesaj aliniyor");
    }

    public void mesajGonder(string message) {
      Console.WriteLine("normal kullanici icin " + message + " gonderiliyor");
    }

    public void accept(Visitor visitor) {
      visitor.visit(this);
    }
  }

  class GoldKullanici : IKullaniciMesajIslem {
    public void kisiBul(string kisi) {
```

```
      Console.WriteLine("gold kullanici icin " + kisi + " adli kisi bulundu");
    }

    public void mesajAl() {
      Console.WriteLine("gold kullanici icin mesaj aliniyor");
    }

    public void mesajGonder(string message) {
      Console.WriteLine("gold kullanici icin " + message + " gonderiliyor");
    }

    public void accept(Visitor visitor) {
      visitor.visit(this);
    }
  }

  class PremiumKullanici : IKullaniciMesajIslem {
    public void kisiBul(string kisi) {
      Console.WriteLine("premium kullanici icin " + kisi + " adli kisi bulundu");
    }

    public void mesajAl() {
      Console.WriteLine("premium kullanici icin mesaj aliniyor");
    }

    public void mesajGonder(string message) {
      Console.WriteLine("premium kullanici icin " + message + " gonderiliyor");
    }

    public void accept(Visitor visitor) {
      visitor.visit(this);
    }
  }

  // Visitor
  interface Visitor {
    void visit(NormalKullanici normalKullanici);
    void visit(GoldKullanici goldKullanici);
    void visit(PremiumKullanici premiumKullanici);
  }

  // Concrete Visitor -> Somut visitor
  class HikayeGosterici : Visitor {
    public void visit(NormalKullanici normalKullanici) {
      Console.WriteLine("Normal kullanici icin hikaye gosteriliyor");
    }

    public void visit(GoldKullanici goldKullanici) {
      Console.WriteLine("Gold kullanici icin hikaye gosteriliyor");
    }

    public void visit(PremiumKullanici premiumKullanici) {
      Console.WriteLine("Premium kullanici icin hikaye gosteriliyor");
    }
  }
```

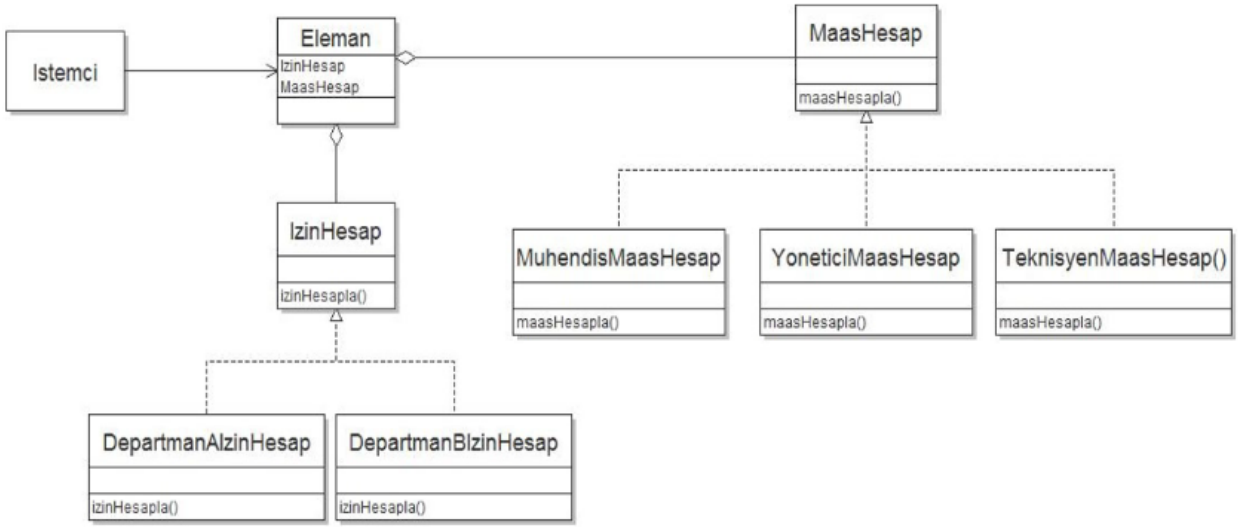
```
  class MesajKaydedici : Visitor {
    public void visit(NormalKullanici normalKullanici) {
      Console.WriteLine("Normal kullanici icin mesaj kaydetme yetkisi yok.");
    }

    public void visit(GoldKullanici goldKullanici) {
      Console.WriteLine("Gold kullanici icin mesaj kaydediliyor");
    }

    public void visit(PremiumKullanici premiumKullanici) {
      Console.WriteLine("Premium kullanici icin mesaj kaydediliyor");
    }
  }

  class Program {
    static void Main(string[] args) {
      Visitor visitor = new HikayeGosterici();
      IKullaniciMesajIslem normalKullanici = new NormalKullanici();
      normalKullanici.accept(visitor);

      visitor = new MesajKaydedici();
      normalKullanici.accept(visitor);

      Console.ReadKey();
    }
  }
}
```

4. Strategy

- Bu tasarım deseninin kullanılmasıyla, kod uzun "if/else" veya "switch" ifadelerinden ayıklanır.
- Geliştirdiğimiz uygulama içerisinde algoritmaları sınıflandırmamızı ve çalışma anında kullanacağımız algoritmayı seçmemizi sağlar.
- Örneğin bir maliyet hesabında LIFO mu yoksa FIFO mu kullanacağını çalışma anında belirlemek istiyorsak bu deseni kullanabiliriz.
- Problem:

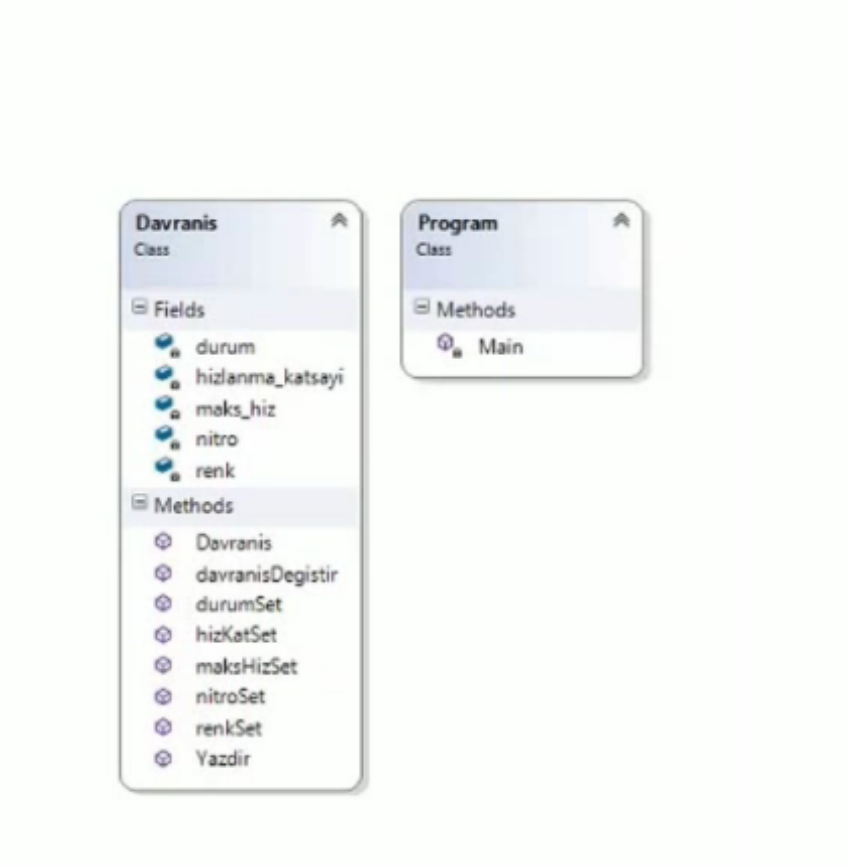


- Çözüm:

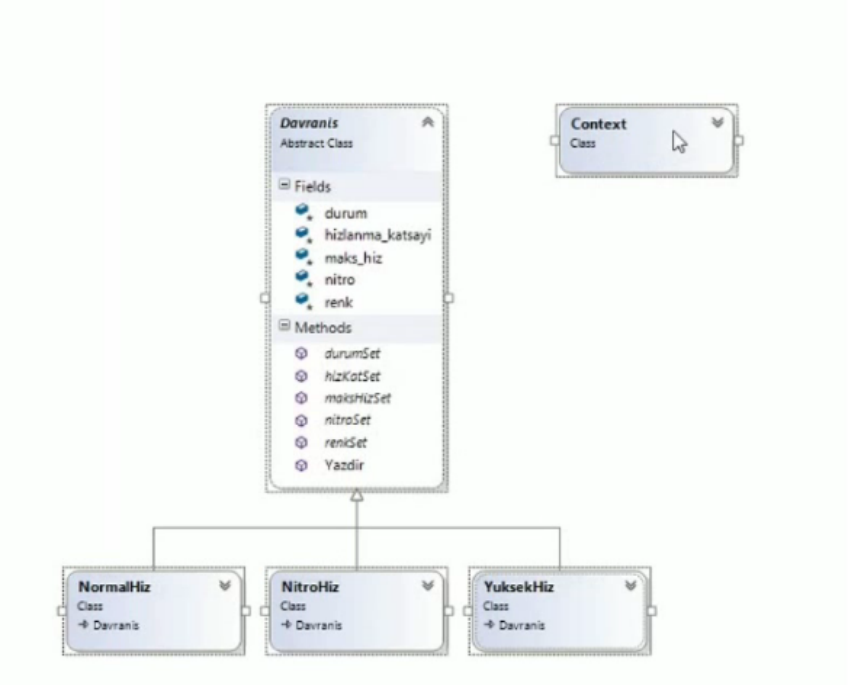


Strategy Uygulaması

- Strategy desensiz:



- Strategy desenli:



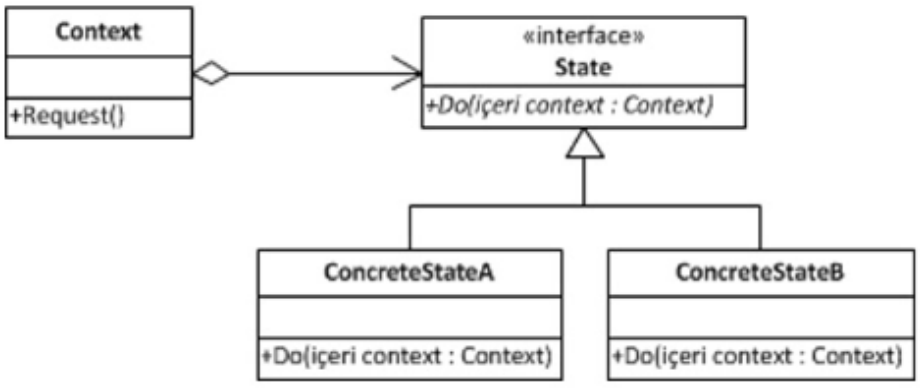
```
namespace Strategy_Pattern {
  abstract class Davranis{
    protected int durum = 0;
    protected int hizlanmaKatsayisi = 1;
    protected int maksimumHiz = 200;
    protected string renk = "beyaz";
    protected bool nitro = false;

    public abstract void durumSet();
    public abstract void renkSet();
    public abstract void hizlanmaKatsayisiSet();
    public abstract void maksimumHizSet();
    public abstract void nitroSet();

    public void yazdir() {
      Console.WriteLine(DateTime.Now);
      Console.WriteLine("Durum: " + durum);
      Console.WriteLine("Renk: " + renk);
      Console.WriteLine("Hizlanma Katsayisi: " + hizlanmaKatsayisi);
      Console.WriteLine("Maksimum Hiz: " + maksimumHiz);
      Console.WriteLine("Nitro: " + nitro);
    }
  }

  class NormalHiz : Davranis {
    public override void durumSet() {
      durum = 0;
    }

    public override void renkSet() {
      renk = "sari";
    }

    public override void hizlanmaKatsayisiSet() {
      hizlanmaKatsayisi = 1;
    }

    public override void maksimumHizSet() {
      maksimumHiz = 200;
    }

    public override void nitroSet() {
      nitro = false;
    }
  }

  class YuksekHiz : Davranis {
    public override void durumSet() {
      durum = 1;
    }

    public override void renkSet() {
      renk = "turuncu";
```

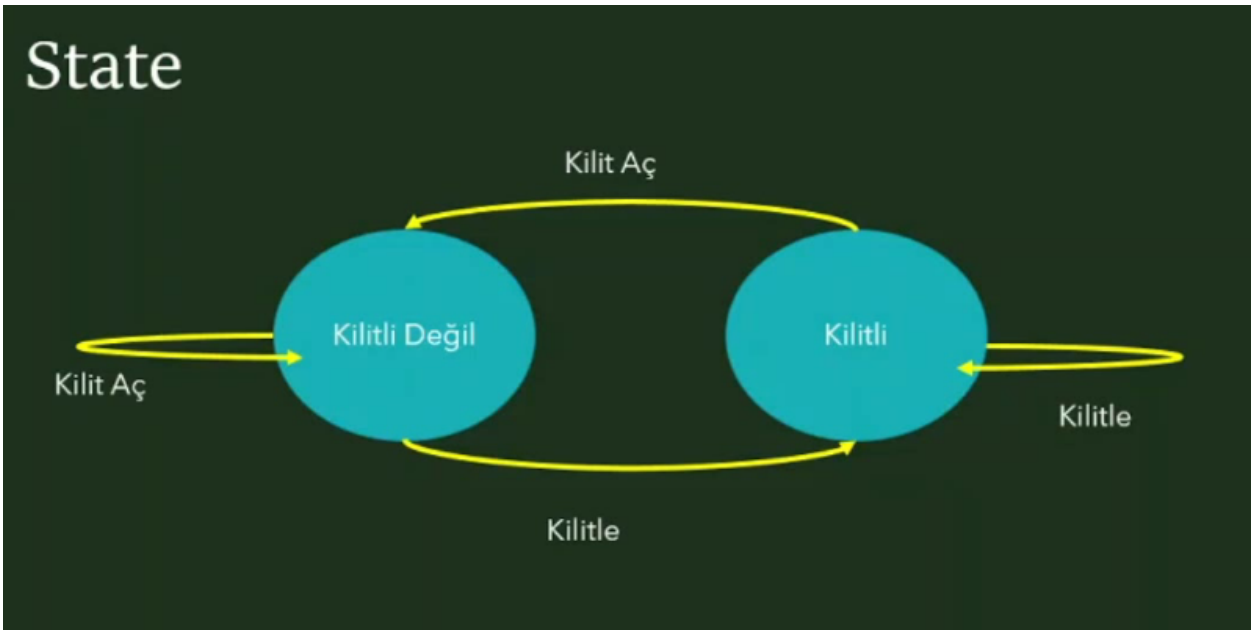
```
    }

    public override void hizlanmaKatsayisiSet() {
      hizlanmaKatsayisi = 2;
    }

    public override void maksimumHizSet() {
      maksimumHiz = 400;
    }

    public override void nitroSet() {
      nitro = false;
    }
  }

  class NitroHiz : Davranis {
    public override void durumSet() {
      durum = 2;
    }

    public override void renkSet() {
      renk = "kirmizi";
    }

    public override void hizlanmaKatsayisiSet() {
      hizlanmaKatsayisi = 3;
    }

    public override void maksimumHizSet() {
      maksimumHiz = 600;
    }

    public override void nitroSet() {
      nitro = true;
    }
  }

  class Context {
    Davranis davranis;

    public void setDavranis(Davranis davranis) {
      this.davranis = davranis;
    }

    public void davranisCalistir() {
      davranis.durumSet();
      davranis.hizlanmaKatsayisiSet();
      davranis.renkSet();
      davranis.maksimumHizSet();
      davranis.nitroSet();
      davranis.yazdir();
    }
  }

  class Program {
```

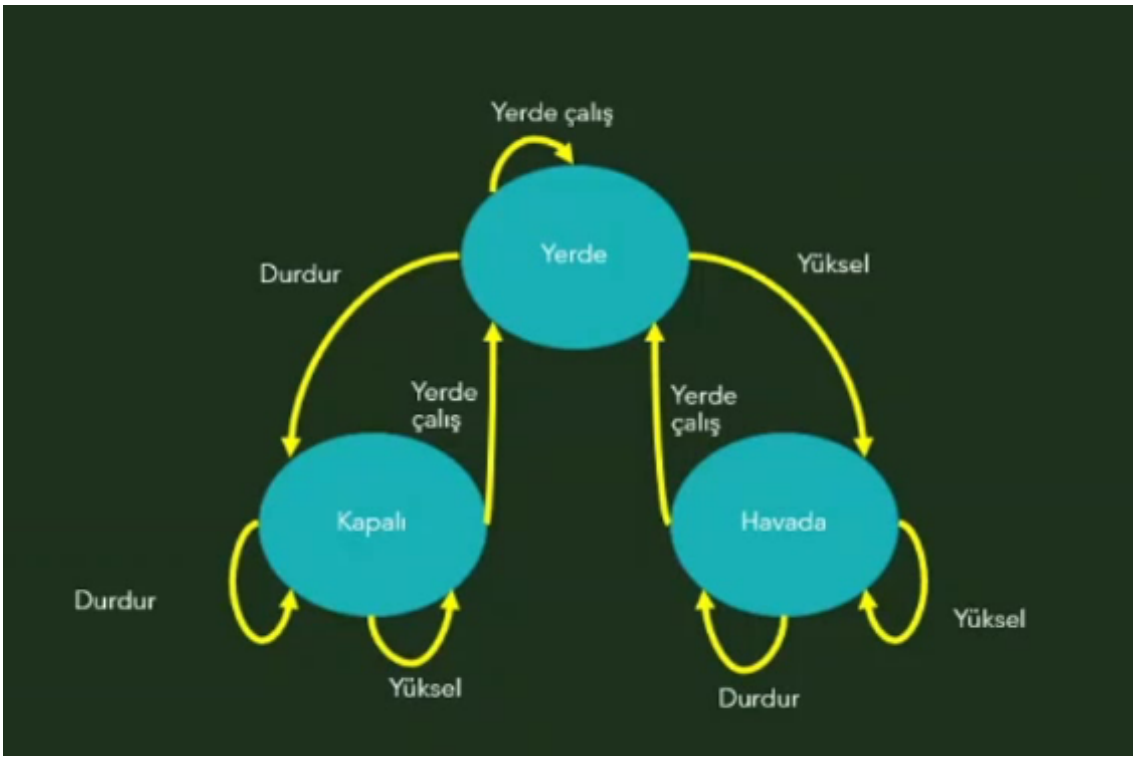
```
    static void Main(string[] args) {

      Context context = new Context();

      Console.WriteLine("Arac normal hizda ilerliyor");
      context.setDavranis(new NormalHiz());
      context.calistir();

      System.Threading.Thread.Sleep(1000);

      Console.WriteLine("Arac yuksek hizda ilerliyor");
      context.setDavranis(new YuksekHiz());
      context.calistir();

      System.Threading.Thread.Sleep(1500);

      Console.WriteLine("Arac nitro hizda ilerliyor");
      context.setDavranis(new NitroHiz());
      context.calistir();

      Console.ReadKey();
    }
  }
}
```

5. State

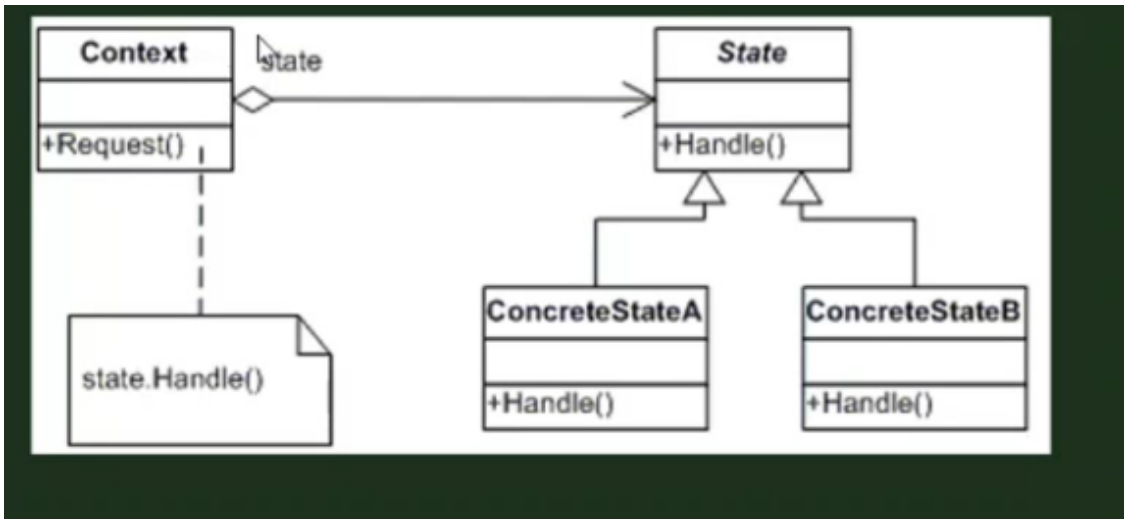
- Nesnelerin farklı durumlarda farklı çalışmalarını düzenler.
- State tasarım deseni; bir nesnedeki bir özellik değiştirildiğinde o nesnenin çalışmasını değiştirmesini veya o durum için kod işlemesini düzenler.







State Uygulaması




```
namespace StateDeseni {
  interface IDurum {
    void durdur();
    void yerdeCalis();
    void yuksel();
  }

  class KapaliDurum : IDurum {
    Context context;

    public KapaliDurum(Context context) {
      this.context = context;
    }

    public void durdur() {
      Console.WriteLine("Drone zaten kapali");
    }

    public void yerdeCalis() {
      Console.WriteLine("Drone yerde calismaya basladi");
      // State degisimi
      context.durumSetle(context.yerdeState);
    }

    public void yuksel() {
      Console.WriteLine("Drone'un oncelikle yerde calismasi gerekir");
    }
  }

  class YerdeDurum : IDurum {
    Context context;

    public YerdeDurum(Context context) {
      this.context = context;
    }
```

```
    public void durdur() {
      Console.WriteLine("Drone durduruldu");
      // State degisimi
      context.durumSetle(context.kapaliState);
    }

    public void yerdeCalis() {
      Console.WriteLine("Drone zaten yerde calisiyor");
    }

    public void yuksel() {
      Console.WriteLine("Drone'un yukseliyor");
      // State degisimi
      context.durumSetle(context.havadaState);
    }
  }

  class HavadaDurum : IDurum {
    Context context;

    public HavadaDurum(Context context) {
      this.context = context;
    }

    public void durdur() {
      Console.WriteLine("Drone'u durdurmak icin yere indirmek gerekir");
    }

    public void yerdeCalis() {
      Console.WriteLine("Drone yerde calisiyor");
      // State degisimi
      context.durumSetle(context.yerdeState);
    }

    public void yuksel() {
      Console.WriteLine("Drone zaten havada");
    }
  }

  class Context {
    public IDurum kapaliState;
    public IDurum yerdeState;
    public IDurum havadaState;

    IDurum simdiki;

    public Context() {
      kapaliState = new KapaliDurum(this);
      yerdeState = new YerdeDurum(this);
      havadaState = new HavadaDurum(this);
      simdiki = kapaliState;
    }

    public IDurum durumGetir() {
```

```
      return simdiki;
    }

    public void durumSetle(IDurum durum) {
      simdiki = durum;
    }

    public void durdurulsun() {
      simdiki.durdur();
    }

    public void yerdeCalissin() {
      simdiki.yerdeCalis();
    }

    public void yukselsin() {
      simdiki.yuksel();
    }
  }

  class Program {
    static void Main(string[] args) {
      Context context = new Context();

      context.yukselsin();
      context.durdurulsun();
      context.yerdeCalissin();
      context.yukselsin();
      context.durdurulsun();
      context.yerdeCalissin();

      Console.ReadKey();
    }
  }
}
```