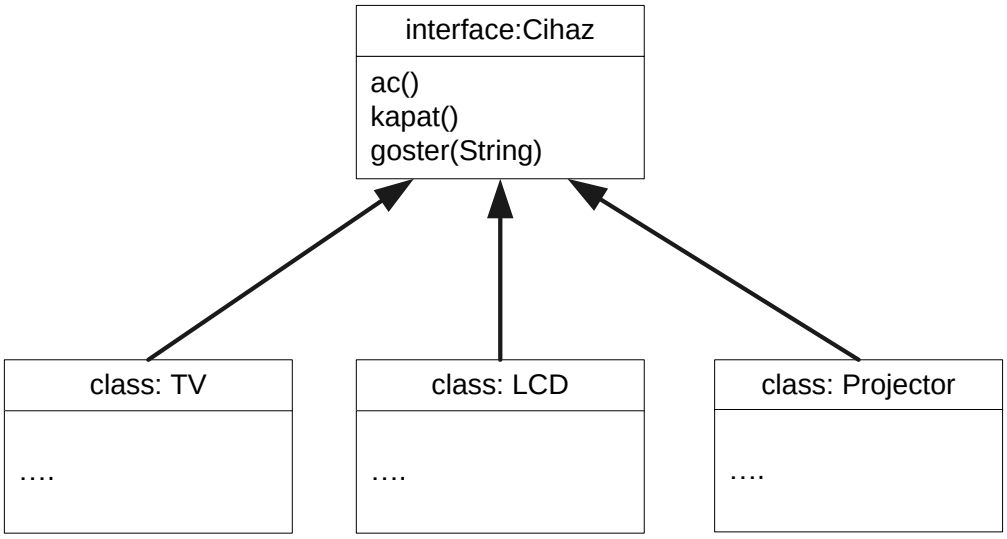
Amaçlar:

1- Arayüzler (Interfaces)
2- Soyut Sınıflar ve Metotlar(Abstract classes and methods)

Uygulama-1:



Cihaz adındaki yukarıda gösterilen Interface'ten türetilmiş **TV, LCD** ve **Projector** adında sınıfları yazınız.
Ayrıca **YayınYap** adında başka bir sınıfta TV, LCD ve Projector sınıflarının nesnelerini üretip bu nesneleri bir Cihaz dizisine atayınız. Daha sonra sırasıyla bu cihazların aç, göster ve kapat metodlarını çağırınız. Herbir cihaz metodları içerisinde kendine özgü mesajlar yazdırmalıdır.

CEVAP

```
public class YayinYap{

    public static void yayinYap(Cihaz[] cihazlar){
        for (int i = 0; i < cihazlar.length; i++) {
            Cihaz cihaz = cihazlar[i];
            cihaz.ac();
            cihaz.goster(" TEST YAYINI");
            cihaz.kapat();
        }
    }
    public static void main(String[] args) {
        Cihaz c1 = new TV();
        Cihaz c2 = new LCD();
        Cihaz c3 = new Projector();
        Cihaz[] dizi = new Cihaz[3];
        dizi[0] = c1;
        dizi[1] = c2;
        dizi[2] = c3;

        YayinYap.yayinYap(dizi);
    }
}
class TV implements Cihaz{
    public void ac(){
        System.out.println("TV Acildi");
    }
    public void kapat(){
        System.out.println("TV Kapandi");
    }
    public void goster(String yazi){
        System.out.println("TV gosteriyor" + yazi);
```

```
    }
}

class LCD implements Cihaz{
    public void ac(){
        System.out.println("LCD Acildi");
    }
    public void kapat(){
        System.out.println("LCD Kapandi");
    }
    public void goster(String yazi){
        System.out.println("LCD gosteriyor" + yazi);
    }
}

class Projector implements Cihaz{
    public void ac(){
        System.out.println("Projector Acildi");
    }
    public void kapat(){
        System.out.println("Projector Kapandi");
    }
    public void goster(String yazi){
        System.out.println("Projector gosteriyor" + yazi);
    }
}
```

Uygulama-2:Aşağıdaki programın çıktısını yazınız:

```
class Class1 {
    public Class1(String isim, int sayi){
        System.out.println("("+isim+","+sayi+")");
    }
}

class Class2 extends Class1{
    private String ad;
    private int no;
    public Class2(String isim, int sayi){
        super("2.SINIF",3);
        this.ad = isim;
        this.no=sayi;
        yazdir();
    }
    public void yazdir(){
        System.out.println("("+this.ad+","+this.no+")");
    }
}

class Class3 extends Class2{
    private String ad;
    public Class3(){
        super("3.SINIF", 5);
        System.out.println("(3,1)");
    }
    public static void main(String[] args) {
        Class3 c = new Class3();
    }
}
```

CEVAP
(2.SINIF,3)
(3.SINIF,5)
(3,1)

Uygulama-3:

notlar1.txt adlı bir dosyanın herbir satırında sırasıyla öğrencinin numarası, 1. Vize notu ve 2. Vize notu yeralmaktadır.

Aşağıda verilen Arayüz (Interface) ve Soyut (Abstract) sınıfları kullanarak NotHesaplayici adında bir sınıf yazınız. Bu sınıf notlar1.txt dosyasından satırları okuyup her bir öğrencinin 1. ve 2. sınavlarının ortalamasını hesaplasın ve sonucu aşağıda verilen şekilde notlar2.txt dosyasına yazsın.

notlar1.txt

```
0801    15.0  25.0
0802    99.0  66.0
0803    60.0  65.0
...
```

notlar2.txt

```
0801    15.0    25.0    20.0
0802    99.0    66.0    82.5
0803    60.0    65.0    62.5
...
```

```
public interface DosyaIslemci {
    public void dosyaOku(String dosyaAdi);
    public void dosyayaYaz(String dosyaAdi, String
dosyaIcerigi);
}

abstract class SatirIslemci{
    public abstract String satirIsle(String satir);
}
```

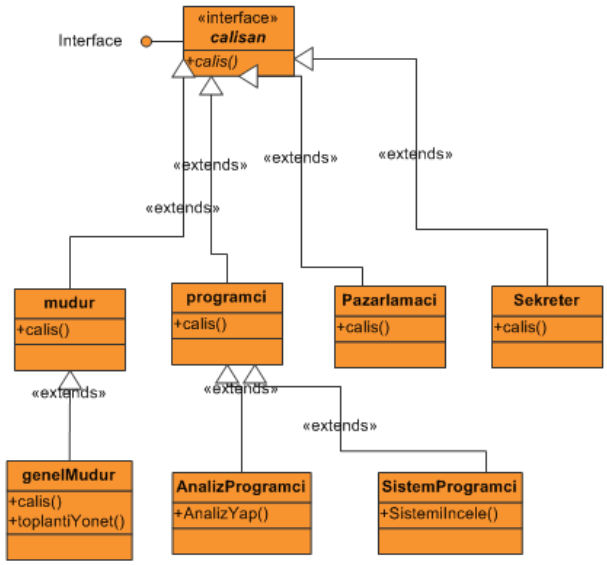
NotHesaplayici sınıfı SatirIslemci sınıfından türemeli ve bu sınıfın main metodunda sadece dosyaOku metodu çağrılmalıdır. satırIsle metodu verilen satırı parcalara böldükten sonra ortalamayı hesaplamalı ve yeni satırı oluşturmalıdır.

CEVAP

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;

class NotHesaplayici extends SatirIslemci implements DosyaIslemci{
   public String satirIsle(String cumle){
      Scanner s = new Scanner(cumle);
       String no = s.next();
       double vize1 = s.nextDouble();
       double vize2 = s.nextDouble();
       double ort = (vize1+vize2)/2;
       String yeniCumle = no + "\t" + vize1 + "\t" + vize2 + "\t" + ort + "\n";
       return yeniCumle;
   }

   public void dosyaOku(String dosyaAdi){
      try {
```

```
            Scanner s = new Scanner(new File(dosyaAdi));
            String yeniDosyaIcerigi = "";
            while(s.hasNext()){
                String satir = s.nextLine();
                yeniDosyaIcerigi += satirIsle(satir);
            }
            System.out.println(yeniDosyaIcerigi);
            dosyayaYaz("notlar2.txt", yeniDosyaIcerigi);
        } catch (Exception e) {}
    }
    public void dosyayaYaz(String dosyaAdi, String dosyaIcerigi){
        try {
            FileWriter fw = new FileWriter(dosyaAdi);
            fw.write(dosyaIcerigi);
            fw.close();
        } catch (Exception e) {}
    }

    public static void main(String[] args) {
        NotHesaplayici n = new NotHesaplayici();
        n.dosyaOku("notlar1.txt");
    }
}
```

Uygulama-4:



```java
package calisan;

interface Calisan{
    public void calis();
}

class mudur implements Calisan{
    @Override
    public void calis(){
        System.out.println("Mudur calisiyor...");
    }
}

class genelMudur extends mudur{
    @Override
    public void calis(){
        System.out.println("Genel mudur calisiyor...");
    }

    public void toplantiYonet(){
        System.out.println("Genel mudur toplanti yonetiyor...");
    }
}

class programci implements Calisan {
    @Override
   public void calis() { // iptal etti (override)
      System.out.println("Programci Calisiyor");
   }
}

class AnalizProgramci extends programci {
   public void analizYap() {
      System.out.println("Analiz Yapiliyor");
   }
```

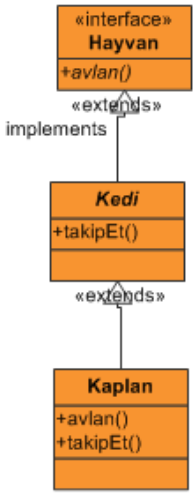
```
}

class SistemProgramci extends programci {
   public void sistemIncele() {
      System.out.println("Sistem Inceleniyor");
   }
}

class Pazarlamaci implements Calisan {
    @Override
   public void calis() { // iptal etti (override)
      System.out.println("Pazarlamaci Calisiyor");
   }
}

class Sekreter implements Calisan {
    @Override
   public void calis() { // iptal etti (override)
      System.out.println("Sekreter Calisiyor");
   }
}

public class BuyukIsYeri {
   public static void mesaiBasla(Calisan[] c ) {
      for (int i = 0 ; i < c.length ; i++) {
         c[i].calis(); // ! Dikkat !
      }
   }
 public static void main(String args[]) {
      Calisan[] c = new Calisan[6];
      // c[0]=new Calisan(); ! Hata ! arayüz olusturulamaz
      c[0]=new programci(); // yukari cevirim (upcasting)
      c[1]=new Pazarlamaci(); //yukari cevirim (upcasting)
      c[2]=new mudur(); //yukari cevirim (upcasting)
      c[3]=new genelMudur(); //yukari cevirim (upcasting)
      c[4]=new AnalizProgramci(); //yukari cevirim (upcasting)
      c[5]=new SistemProgramci(); //yukari cevirim (upcasting)
      mesaiBasla(c);
   }
}
```

Program Çıktısı:

```
Programci Calisiyor
Pazarlamaci Calisiyor
Mudur calisiyor...
Genel mudur calisiyor...
Programci Calisiyor
Programci Calisiyor
```

Not: Çalışan arayüzüne erişen tüm sınıflarda, bu arayüz (interface) içerisinde bulunan gövdesiz metotlar override edilmek zorundadır.

Uygulama-5:

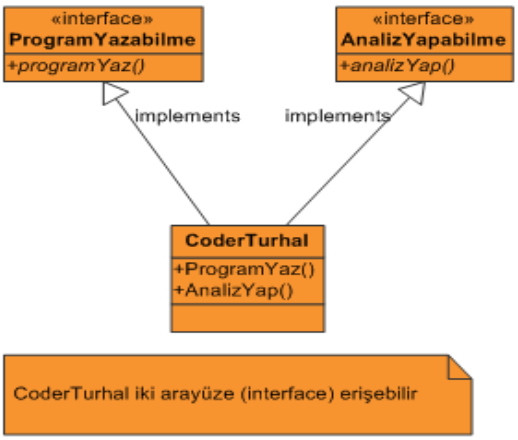


Kaplan sınıfı, tüm gövdesiz metodları (soyut metodları) iptal etmek zorunda

```
package hayvanat;

interface Hayvan {
   public void avlan() ;
}

abstract class Kedi implements Hayvan {
   public abstract void takipEt() ;
}

class Kaplan extends Kedi {
    @Override
   public void avlan() { // iptal etti (override)
      System.out.println("Kaplan avlaniyor...");
   }

    @Override
   public void takipEt() { // iptal etti (override)
      System.out.println("Kaplan takip ediyor...");
   }
}

public class Hayvanat {
    public static void main(String[] args) {
        Kaplan kpln=new Kaplan();
        kpln.avlan();
        kpln.takipEt();
    }
}
```

Not: Kaplan sınıfı kedi soyut sınıfındaki gövdesiz metodu override etmek zorundadır. Ayrıca kedi sınıfı da Hayvan arayüzüne ulaştığı için bu arayüz içindeki metodu da override etmek zorundadır.

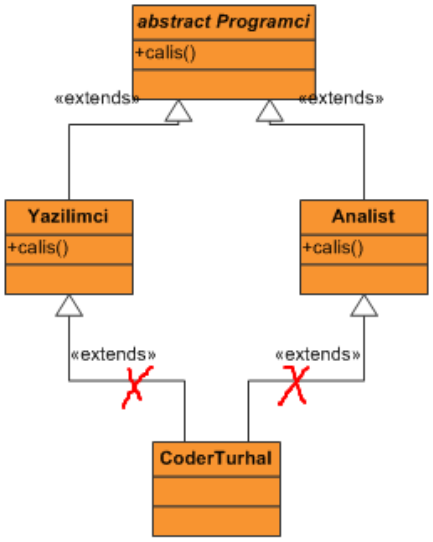
Uygulama-6:




```
package program;

interface ProgramYazabilme {

    public void programYaz();
}

interface AnalizYapabilme {

    public void analizYap();
}

class Deneme implements ProgramYazabilme, AnalizYapabilme {

    @Override
    public void programYaz() {
        System.out.println("Program yaziyor");
    }

    @Override
    public void analizYap() {
        System.out.println("Analiz yapıyor");
    }
}

public class Program {

    public static void main(String[] args) {

        Deneme dene=new Deneme();
        dene.programYaz();
        dene.analizYap();
    }
}
```

Not: Arayüzler ile çoklu kalıtım yapılabilmektedir. Başka bir deyişle aynı bağlantı yolu üzerinde 1'den fazla extends işlemi gerçekleştirilememektedir. Bu durumun önüne arayüzler kullanılarak geçilebilmektedir. Aşağıdaki UML diyagramı bu işlemin neden olmayacağı durumunu özetlemektedir.

Uygulama-6:




```
abstract class Programci {
  public abstract void calis();
}

class Yazilimci extends Programci {
  public void calis() {
    System.out.println("Yazilimci calisiyor....") ;
  }
}

class Analist extends Programci {
  public void calis() {
    System.out.println("Analist calisiyor....") ;
  }
}

/*
Bu ornegimiz derleme aninda hata alicaktir.
Java, coklu kalitimi desteklemez
*/
class CoderTurhal extends Yazilimci, Analist {
}
```