

Diseinu Patroiak

SOFTWARE INGENIARITZA



Portaerazkoak

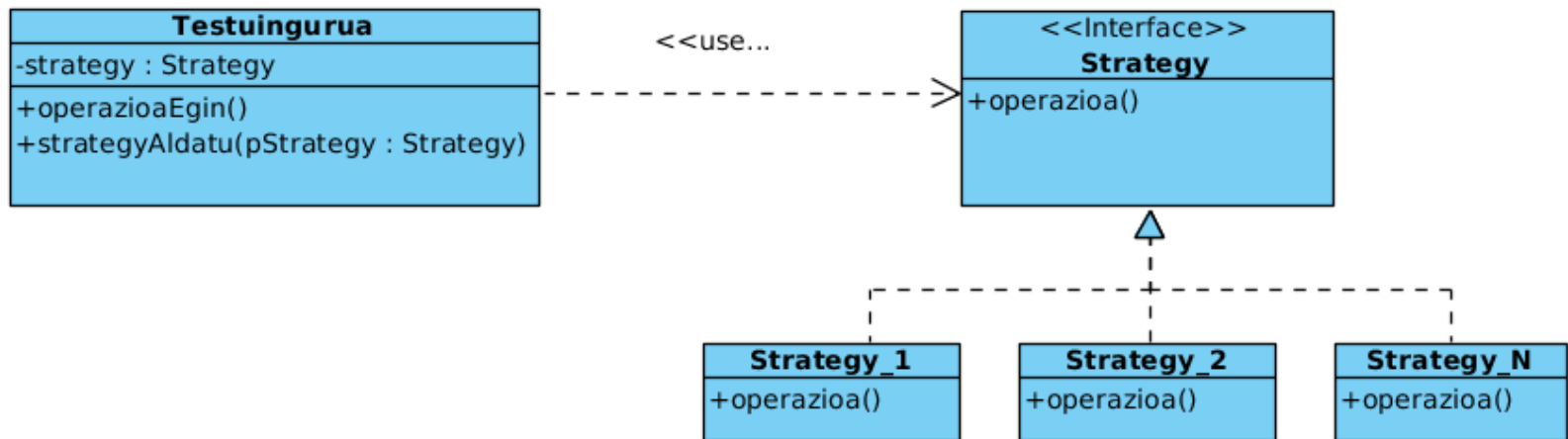


Strategy

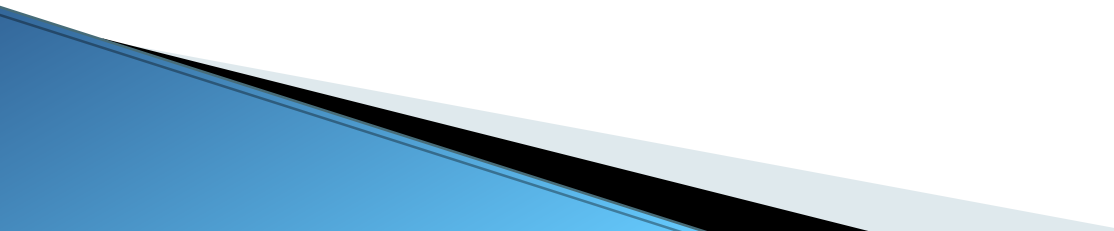


Eskema Orokorra

Strategy: funtzionalitate beraren portaera (estrategia) desberdinak definitzen ditu, eta testuinguruaren arabera estrategia hautatzea ahalbidetzen. Gainera, estrategia “run-time”ean aldatzeko aukera emanten du.



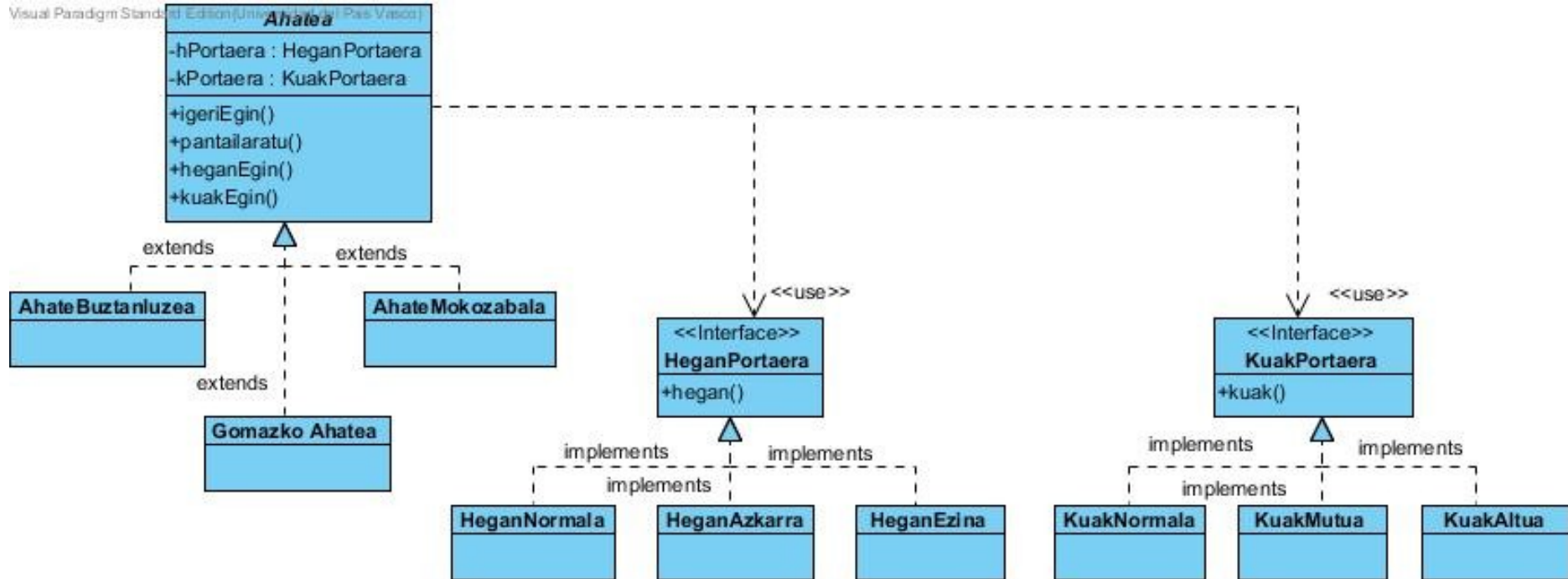
Ezaugarriak

- ▶ Testuingurua eta portaerak banatu
 - ▶ Portaerak aldagarriak dira; klase konkretuetan kapsulatu
 - ▶ Berrerabilpena/hedapena hobetu
 - ▶ Algoritmo familiak (estrategiak) definitu
 - ▶ Bezeroak estrategia aldatu dezake
- 

Arazoa

- ▶ Ahateak simulatzen dituen sistema diseinatu.
 - Ahate motak: *buztanluzea, moko zabala, gomazkoa*
- ▶ Ahateek *hegan* eta *kuak* egin dezakete:
 - Hegan portaerak: *normala, azkarra, ezina*
 - Kuak portaerak: *normala, altua, mutua*
- ▶ Ahate moten *hegan* eta *kuak* portaerak:
 - Buztanluzea: hegan normala eta kuak altua.
 - Moko zabala: hegan azkarra eta kuak normala. Gainera *kuak portaera alda dezake*.
 - Gomazkoa: hegan ezina eta kuak mutua.

Ebazpena



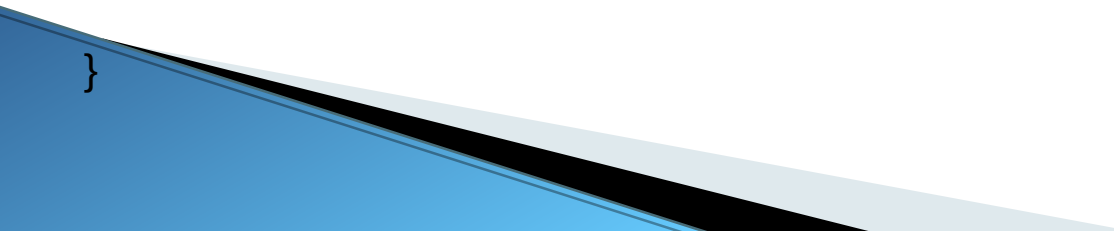
Ebazpena

```
public interface HeganPortaera {  
    public void hegan();  
}
```

```
public class HeganNormala implements HeganPortaera{  
    public HeganNormala(){  
    public void hegan(){System.out.println("Hegan ari naiz!");}  
}
```

```
public class HeganAzkarra implements HeganPortaera{  
    public HeganAzkarra(){  
    public void hegan(){System.out.println("Azkar ari naiz hegan!");}  
}
```

```
public class HeganEzina implements HeganPortaera{  
    public HeganEzina(){  
    public void hegan(){System.out.println("Ezin dut hegan egin!");}  
}
```



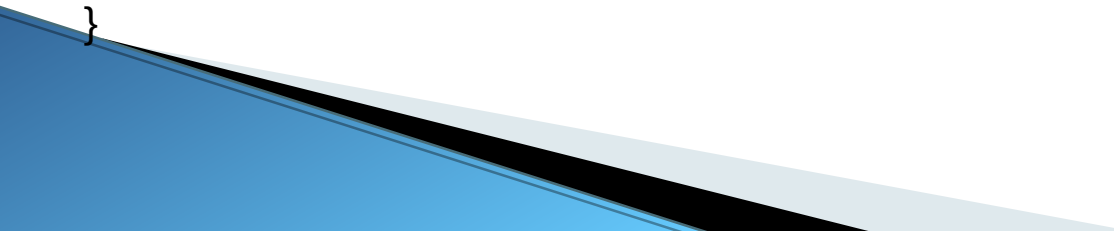
Ebazpena

```
public interface KuakPortaera {  
    public void kuak();  
}
```

```
public class KuakNormala implements KuakPortaera{  
    public KuakNormala(){  
    public void kuak(){System.out.println("Kuak!");}  
}
```

```
public class KuakAltua implements KuakPortaera{  
    public KuakAltua(){  
    public void kuak(){System.out.println("KUAK!");}  
}
```

```
public class KuakMutua implements KuakPortaera{  
    public KuakMutua(){  
    public void kuak(){System.out.println("...!");}  
}
```



Ebazpena

```
public abstract class Ahatea {
```

```
    protected HeganPortaera hPortaera;  
    protected KuakPortaera kPortaera;
```

```
    public Ahatea () {}
```

```
    public void igeriEgin(){System.out.println ("Igerian ari naiz!");}
```

```
    public abstract void pantailaratu();
```

```
    public void heganEgin(){hPortaera.hegan();}
```

```
    public void kuakEgin(){kPortaera.kuak();}
```

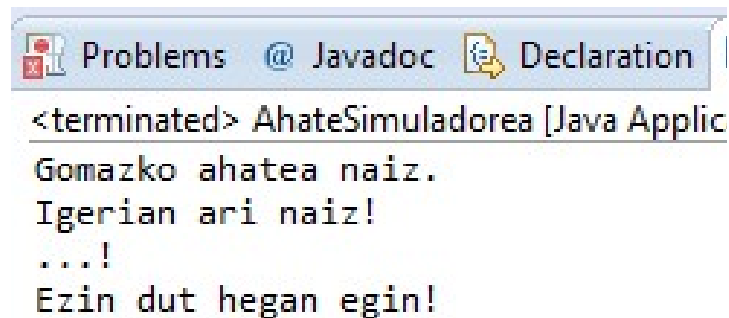
```
}
```

Ebazpena

```
public class GomazkoAhatea extends Ahatea{  
    public GomazkoAhatea(){  
        hPortaera = new HeganEzina();  
        kPortaera = new KuakMutua();  
    }  
    public void pantailaratu(){System.out.println("Gomazko ahatea naiz.");}  
}  
  
public class AhateMokozabala extends Ahatea{  
    public AhateMokozabala(){  
        hPortaera = new HeganAzkarra();  
        kPortaera = new KuakNormala();  
    }  
    public void kuakAldatu(KuakPortaera pKuakPortaera){  
        kPortaera = pKuakPortaera;  
    }  
    public void pantailaratu(){System.out.println("Ahate Mokozabala naiz.");}  
}
```

Ebazpena

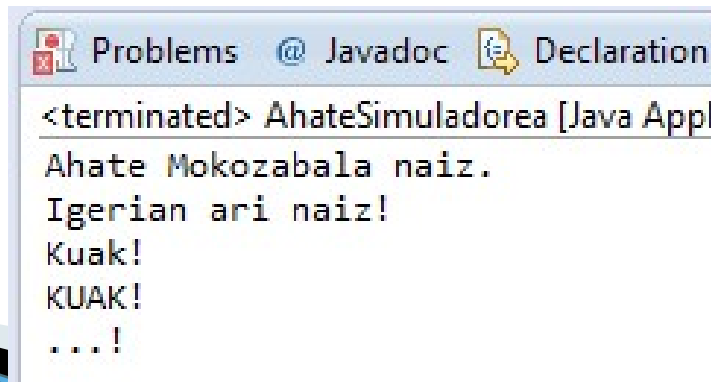
```
public class AhateSimuladorea {  
  
    public static void main(String[] args) {  
  
        Ahatea probaAhatea = new GomazkoAhatea();  
        probaAhatea.pantailaratu();  
        probaAhatea.igeriEgin();  
        probaAhatea.kuakEgin();  
        probaAhatea.heganEgin();  
    }  
}
```



Ebazpena

```
public class AhateSimuladorea {  
  
    public static void main(String[] args) {  
        AhateMokozabala probaAhatea = new AhateMokozabala();  
        probaAhatea.pantailaratu();  
        probaAhatea.igeriEgin();  
        probaAhatea.kuakEgin();  
  
        probaAhatea.kuakAldatu(new KuakAltua());  
        probaAhatea.kuakEgin();  
        probaAhatea.kuakAldatu(new KuakMutua());  
        probaAhatea.kuakEgin();  
    }  
}
```

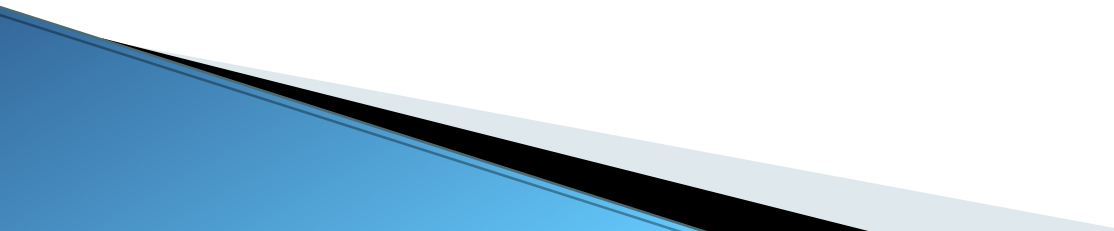
Dinamikoki ari
naiz portaera
aldatzen

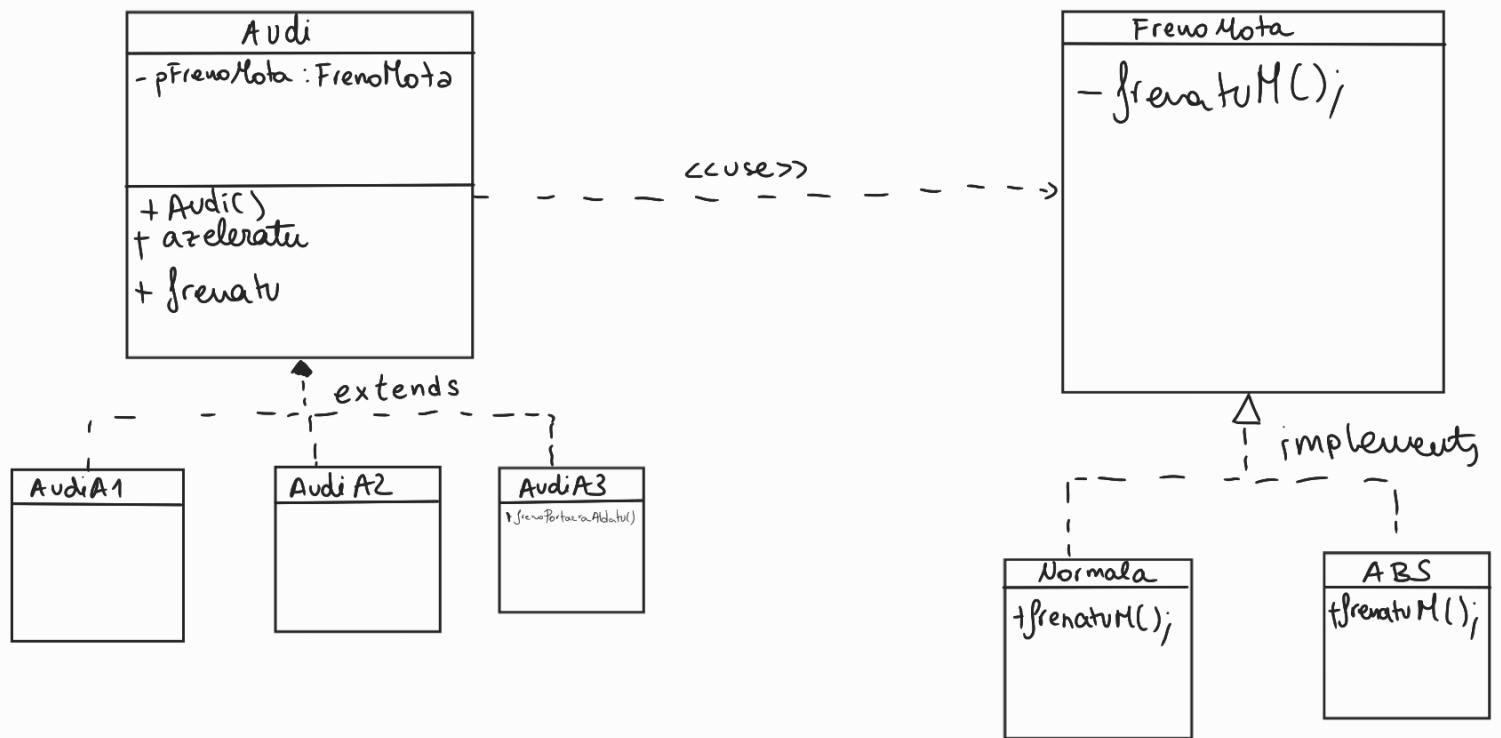


The screenshot shows a Java IDE console window with the following output:

```
<terminated> AhateSimuladorea [Java Appl  
Ahate Mokozabala naiz.  
Igerian ari naiz!  
Kuak!  
KUAK!  
...!
```

Ariketa: Audi Kotxeak

- ▶ Audi kotxeak kontrolatzen dituen sisteman lanean jarraitzen dugu.
 - ▶ Balazta zapaltzean gertatzen dena kontrolatzeko sistema inplementatu.
 - ▶ Bi balazta sistema daude, *normala* eta *ABS*.
 - ▶ Hiru Audi modelo: A1 (ABS) dauka, A2 (Normala) eta A3 (ABS edo normala aukeratu)
 - ▶ Sistemaren diseinua egin (klase diagrama)
- 



```

public abstract class Audi() {
    protected FrenoMota pFrenoMota;
    private Audi() {}
    public void frenatu() {
        pFrenoMota.frenatuM();
    }
}

```

```

public class AudiA1 extends Audi {
    public AudiA1 {
        frenuMota = new ABS();
    }
}

```

```

public class AudiA2 extends Audi {
    public AudiA2 {
        frenuMota = new Normala();
    }
}

```

```
public class AudiA3 extends Audi {  
    public AudiA3 {  
        frenoMota = new ABS();
```

```
}
```

```
public void FrenoPortaeraAldatu(FrenoMota sFrenoMota) {  
    frenoMota = sFrenoMota;
```

```
}
```

```
}
```

```
public interface FrenoMota() {  
    public void frenatuM();
```

```
}
```

```
public class ABS implements FrenoMota() {
```

```
    public ABS() {}
```

```
    public void frenatuM() { System.out.println("Frenatzen ABS"); }
```

```
public class Normala implements FrenoMota() {
```

```
    public Normala() {}
```

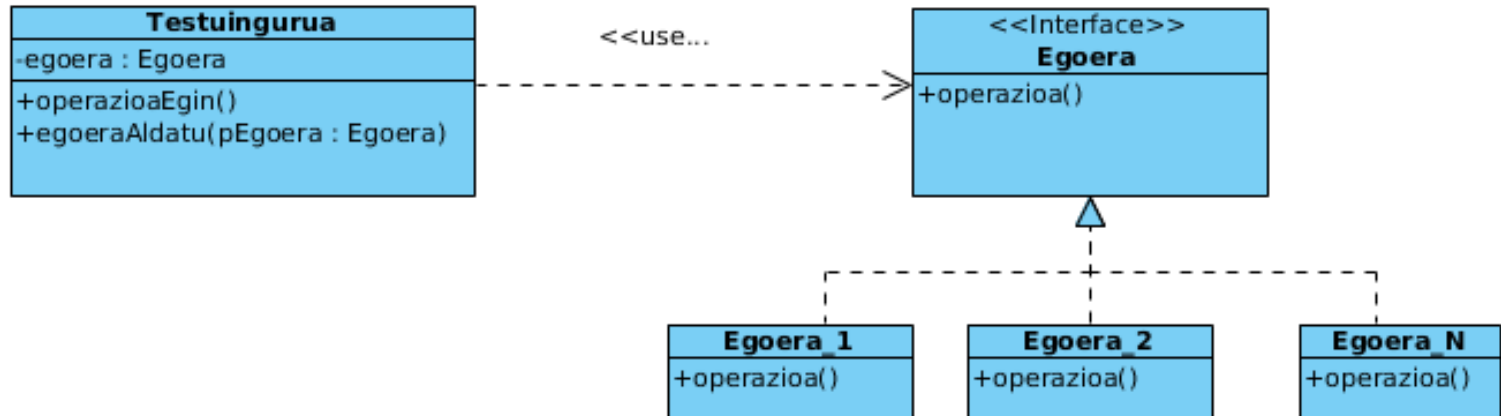
```
    public void frenatuM() { System.out.println("Frenatzen Normal"); }
```


State

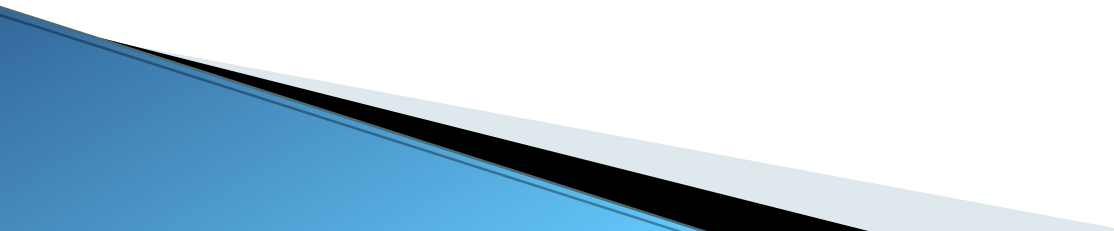


Eskema Orokorra

State: objektu baten barne egoera aldatzean, bere portaera aldatzea ahalbidetzen dio.



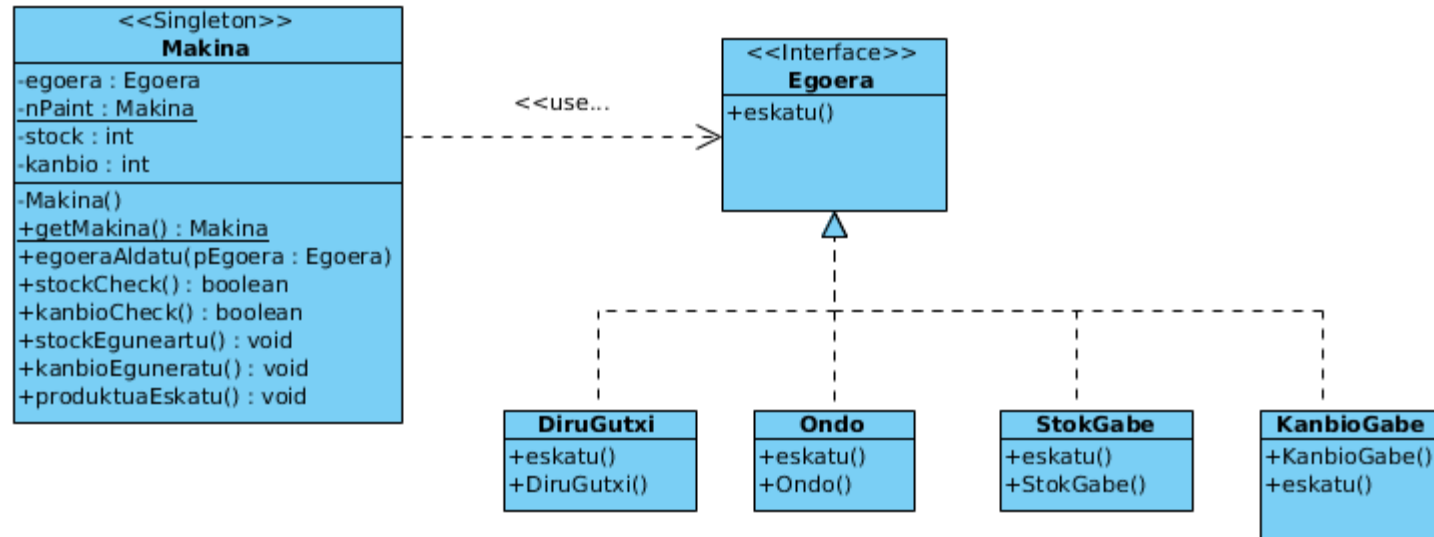
Ondorioak

- ▶ Egoeraren arabera, portaera desberdina
 - ▶ Egoeren portaera klase konkretuek kapsulatu
 - ▶ Egoeren arteko transizio esplizituak
 - ▶ Hedagarria
 - ▶ Bezeroak egoeren informazio gutxi edo ezer
- 

Arazoa

- ▶ Vending makina bat kontrolatzeko sistema. Eskaera botoiari zaplatzean duen portaera diseinatuko dugu.
 - *Ondo* : diru nahikoa, stock-a eta kanbioak daude
→ *produktua eman*
 - *Stock gabe* : diru nahikoa, kanbioak daude, stock-ik ez
→ *errore mezua*
 - *Diru gutxi* : diru gutxiegi sartu, stock-a eta kanbioak daude
→ *diru gehiago eskatu*
 - *Kanbiorik ez* : diru nahiko, stock-a dago, kanbiorik ez
→ *diru zehatza eskatu*
 - Funtzionalitatea diseinatu, makinaren arabera, egoera gehiago egon daitezkeela jakinda.

Ebazpena



```
public class Makina {
```

```
    private Egoera egoera;  
    private int stock = 2;  
    private int kanbioa = 1;  
    private static Makina nMakina;
```

```
    private Makina(){egoera = new Ondo();}
```

```
    public static Makina getMakina(){  
        if (nMakina == null) {nMakina = new Makina();}  
        return nMakina;}  
}
```

```
    public void egoeraAldatu(Egoera pEgoera){egoera = pEgoera;}
```

```
    public void produktuaEskatu(){egoera.eskatu();}
```

```
    //Stock eguneraketaren sinplifikazio bat
```

```
    public void stockEguneratu(){stock--;}
```

```
    //Kanbio kudeaketaren sinplifikazio bat
```

```
    public void kanbioEguneratu(){kanbioa--;}
```

```
    public boolean stockCheck(){return stock == 0;}
```

```
    public boolean kanbioCheck(){return kanbioa == 0;}
```

```
}
```

Erabiltzaileak hau bakarrik egin dezake. Stock eta kanbioen arabera, egoera desberdin batera joango da, eta portaera desberdina izango du


Ebazpena

```
public interface Egoera {  
    public void eskatu();  
}  
public class Ondo implements Egoera{  
    public Ondo(){}  
    public void eskatu(){  
        System.out.println("--> Produktua emango du.");  
        Makina.getMakina().stockEguneratu();  
        Makina.getMakina().kanbioEguneratu();  
        if(Makina.getMakina().stockCheck())  
            Makina.getMakina().egoeraAldatu(new StockGabe());  
        else if(Makina.getMakina().kanbioCheck())  
            Makina.getMakina().egoeraAldatu(new KanbioGabe());  
    }  
}  
public class StockGabe implements Egoera{  
    public StockGabe(){}  
    public void eskatu(){System.out.println("-->Produktua ez dago stock-ean.");}  
}
```

Ebazpena

```
public class DiruGutxi implements Egoera{
    public DiruGutxi(){}
    public void eskatu(){
        System.out.println("--> Ez duzu diru nahikoa sartu.");
        //Ondo sartzen duenean
        Makina.getMakina().egoeraAldatu(new Ondo());
    }
}

public class KanbioGabe implements Egoera{
    public KanbioGabe(){}
    public void eskatu(){
        System.out.println("--> Mesedez, diru zehatza sartu.");
        //Diru zehatza sartzen badu
        System.out.println("--> Diru zehatza bada produktua eman.");
        Makina.getMakina().stockaEguneratu();
        if(Makina.getMakina().stockGabe())
            Makina.getMakina().egoeraAldatu(new StockGabe());
    }
}
```



Ebazpena

```
public class MakinaSimuladorea {
```

```
    public static void main(String[] args) {
```

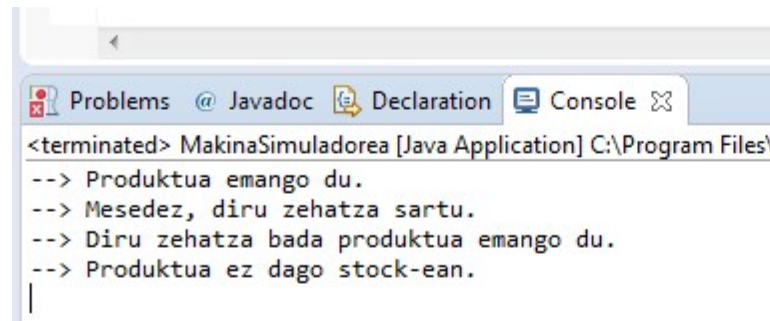
```
        Makina.getMakina().produktuaEskatu(); //1- "Ondo" egoeratik habiatu  
                                                // Stock/Kanbio eguneratu  
                                                // "Ondo" -> "KanbioGabe"
```

```
        Makina.getMakina().produktuaEskatu(); //2- "KanbioGabe" egoeran  
                                                // Stock/Kanbio eguneratu  
                                                // "KanbioGabe" -> "Stockgabe"
```

```
        Makina.getMakina().produktuaEskatu(); //3- "StockGabe" egoeran
```

```
    }
```

```
}
```

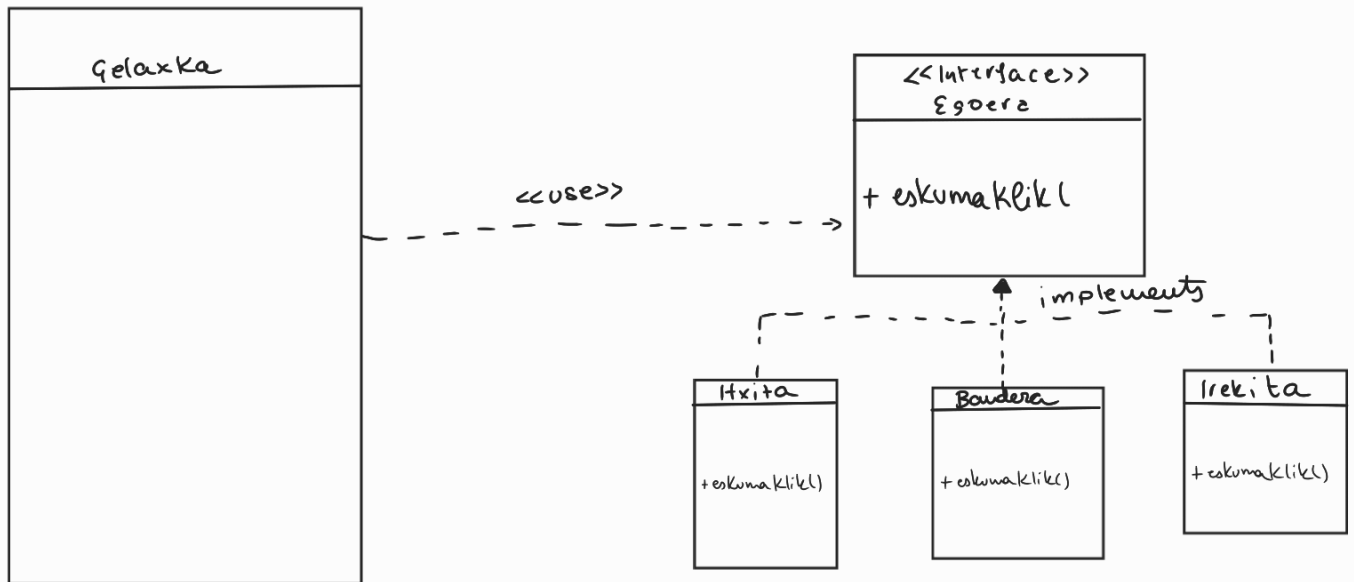


```
<terminated> MakinaSimuladorea [Java Application] C:\Program Files\  
--> Produktua emango du.  
--> Mesedez, diru zehatza sartu.  
--> Diru zehatza bada produktua emango du.  
--> Produktua ez dago stock-ean.  
|
```

Ariketa: Dragamina

- ▶ Dragamina jokoko panelean, gelaxka baten eskubiko klika simulatu:
 - Gelaxka itxita badago, bandera irudia jarri eta bandera egoerara pasatu.
 - Gelaxka banderarekin badago, bandera irudia kendu eta gelaxka itxi.
 - Gelaxka irekita badago, ez du ezer egiten.

STATE



STATE vs. STRATEGY

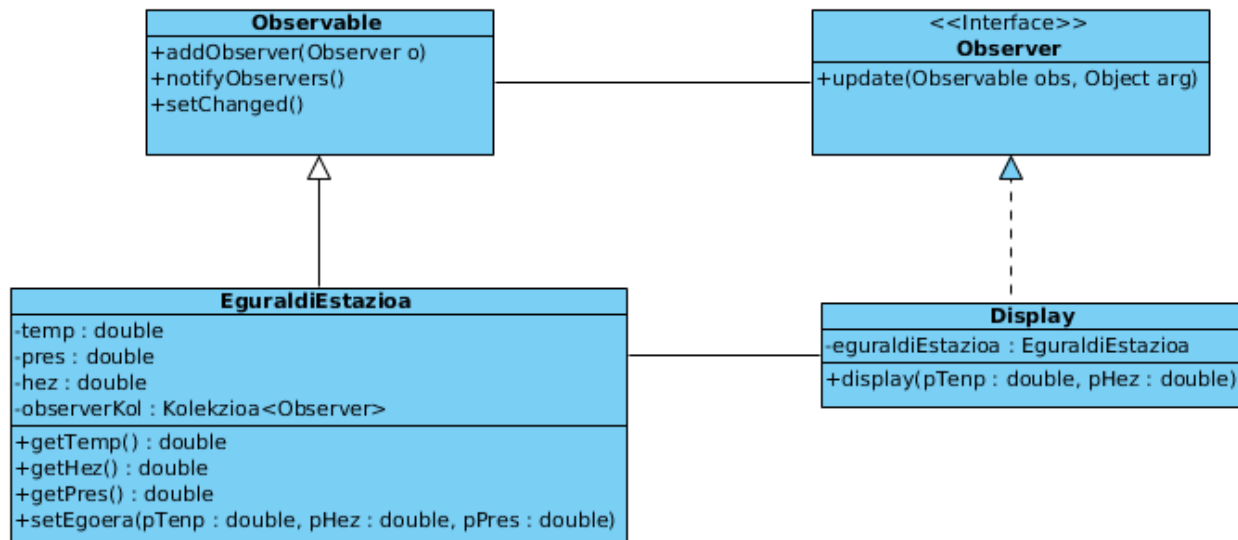
- ▶ Eskema bera dute, zein da beraien arteko desberdintasuna?
 - Testuinguru/portaera
 - Strategy: hainbat testuinguru, eta bakoitzak portaera desberdinak
 - State: testuinguru bakarra, hainbat egoerarekin
 - Erabiltzailearen ikuspuntua
 - Strategy: estrategiak ezagutu ditzake, baita aukeratu ere
 - State: ez daki barne egoeren inguruan ia ezer edo ezer ez. Ezin du egoera aldatzeko erabakirik hartu, dagokion operazioa bakarrik burutu dezake.

Observer

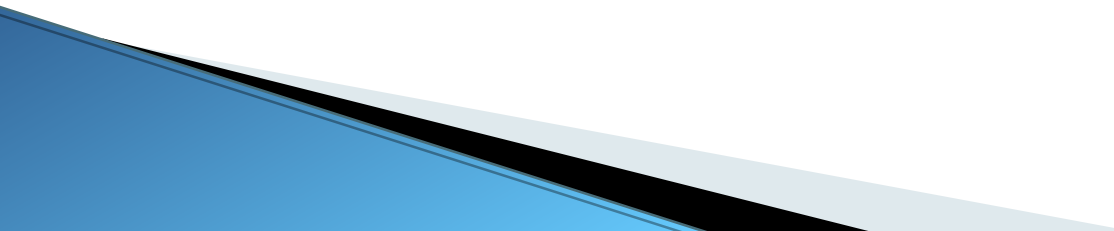


Eskema orokorra

Observer: objektuen behagarri bat (*observable*) eta azken hori behatzen dutenak (*observers*) arteko dependentziak definitzeko balio du; *observable*-k bere egoera aldatzen duenean, *observer* guztiei jakinaraziko die.

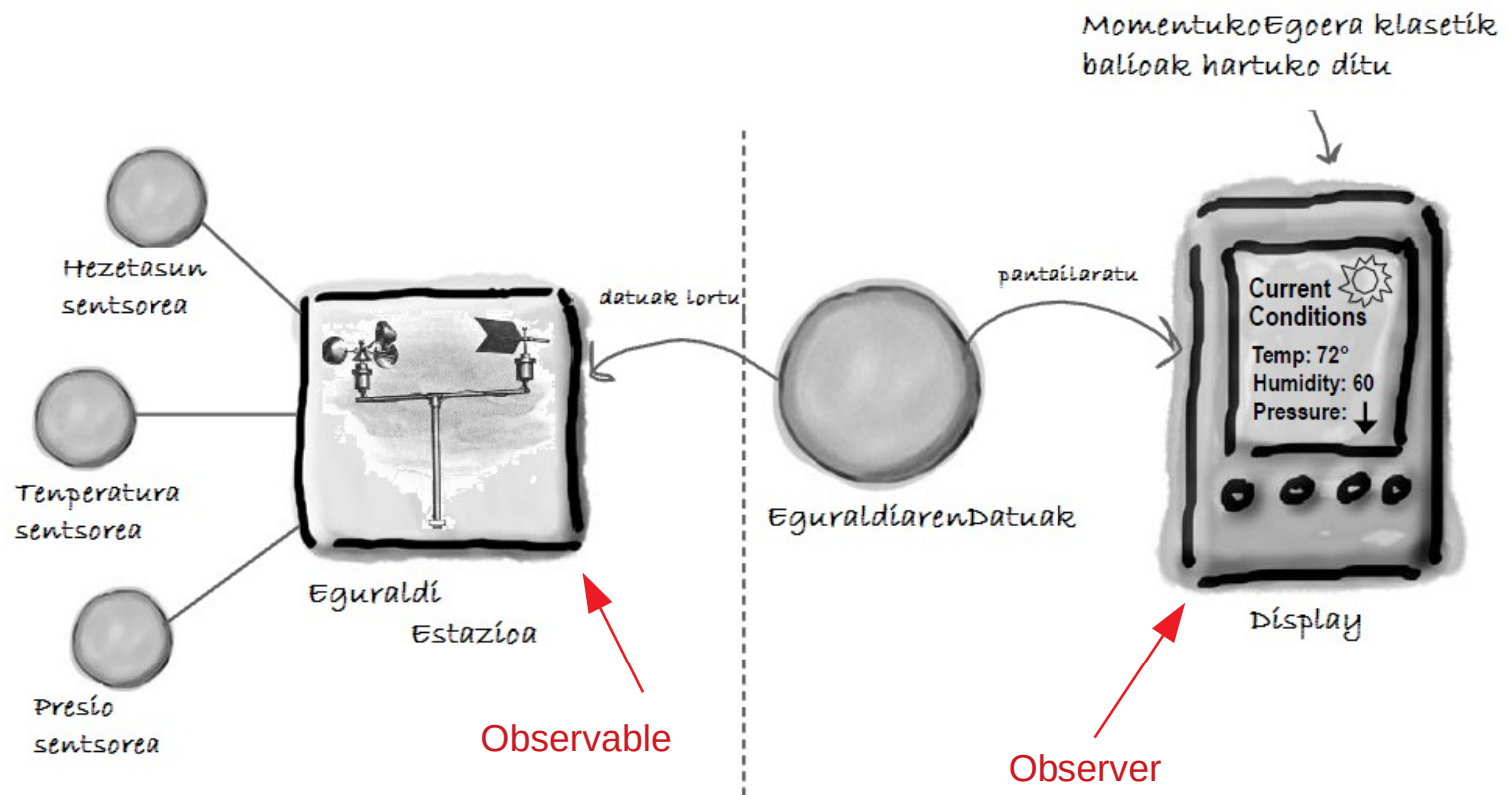


Ondorioak

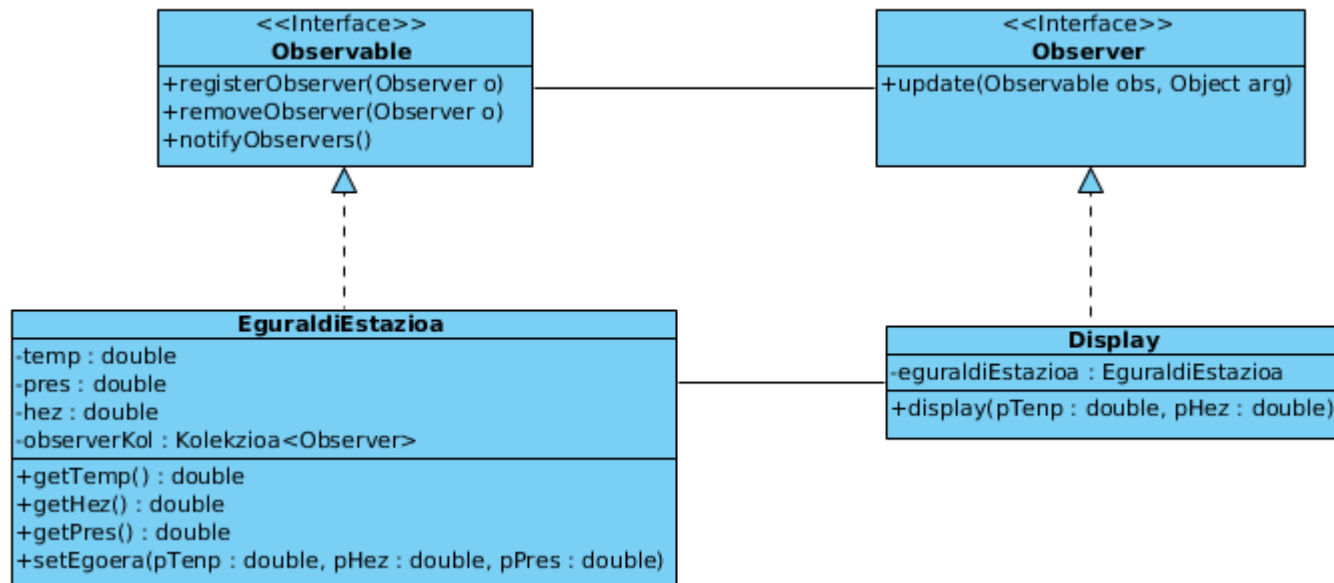
- ▶ Akoplamendu soltea (loose-coupling); *observable*-ak ez daki *observer* motaren inguruan.
 - ▶ Entzuleei jakinarazpenak bidali
 - ▶ Hedagarria
- 

Arazoa

- Sistemaren klase diagrama egin, gutxienez bi display egon daitezkeela jakinda.



Ebazpena



Ebazpena: Java

```
public class EguraldiEstazioa implements Observable{
```

```
    private ArrayList observerKol;  
    private double tenp;  
    private double hez;  
    private double pres;
```

Observer kolekzioa

```
    public EguraldiEstazioa() {  
        observerKol = new ArrayList<>();  
    }
```

```
    public void registerObserver(Observer o) {  
        observerKol.add(o);  
    }
```

```
    public void removeObserver(Observer o) {  
        int i = observerKol.indexOf(o);  
        if (i >= 0) {  
            observerKol.remove(i);  
        }  
    }
```

Observable-n
egoera aldaketa
bat simulatzeko

```
    public void setEgoera(double pTenp, double pHez, double pPres)  
    { //Eguraldian aldaketak egon dira!  
        this.tenp = pTenp; this.hez = pHez; this.pres = pPres;  
        egoeraAldatuDa();  
    }
```

Ebazpena: Java

```
public void egoeraAldatuDa() {  
    notifyObservers();  
}  
  
public void notifyObservers() {  
    for (int i = 0; i < observerKol.size(); i++) {  
        Observer observer = (Observer)observerKol.get(i);  
        observer.update(this);  
    }  
}  
  
public double getTenp() {return tenp;}  
public double getHez() {return hez;}  
public double getPres() {return hez;}  
}
```

Egoera aldaketa
detektatzerakona
Observer-ei
jakinarazpena

Ebazpena: Java

```
public class Display implements Observer{

    private Observable eguraldiEstazioa;

    public Display (Observable pEguraldiEstazioa) {
        eguraldiEstazioa = pEguraldiEstazioa;
        eguraldiEstazioa.registerObserver(this);
    }


    public void update(Observable obs){
        System.out.println("Eguraldi estazioan aldaketaren bat egon da!!");
        display(((EguraldiEstazioa)obs).getTenp(), ((EguraldiEstazioa)obs).getHez());
    }

    public void display(double pTenp, double pHez) {
        System.out.println("Momentuko datuak: " + pTenp + "gradu eta " + pHez + "%
hezetasuna");
    }

}
```

Ebazpena: Java

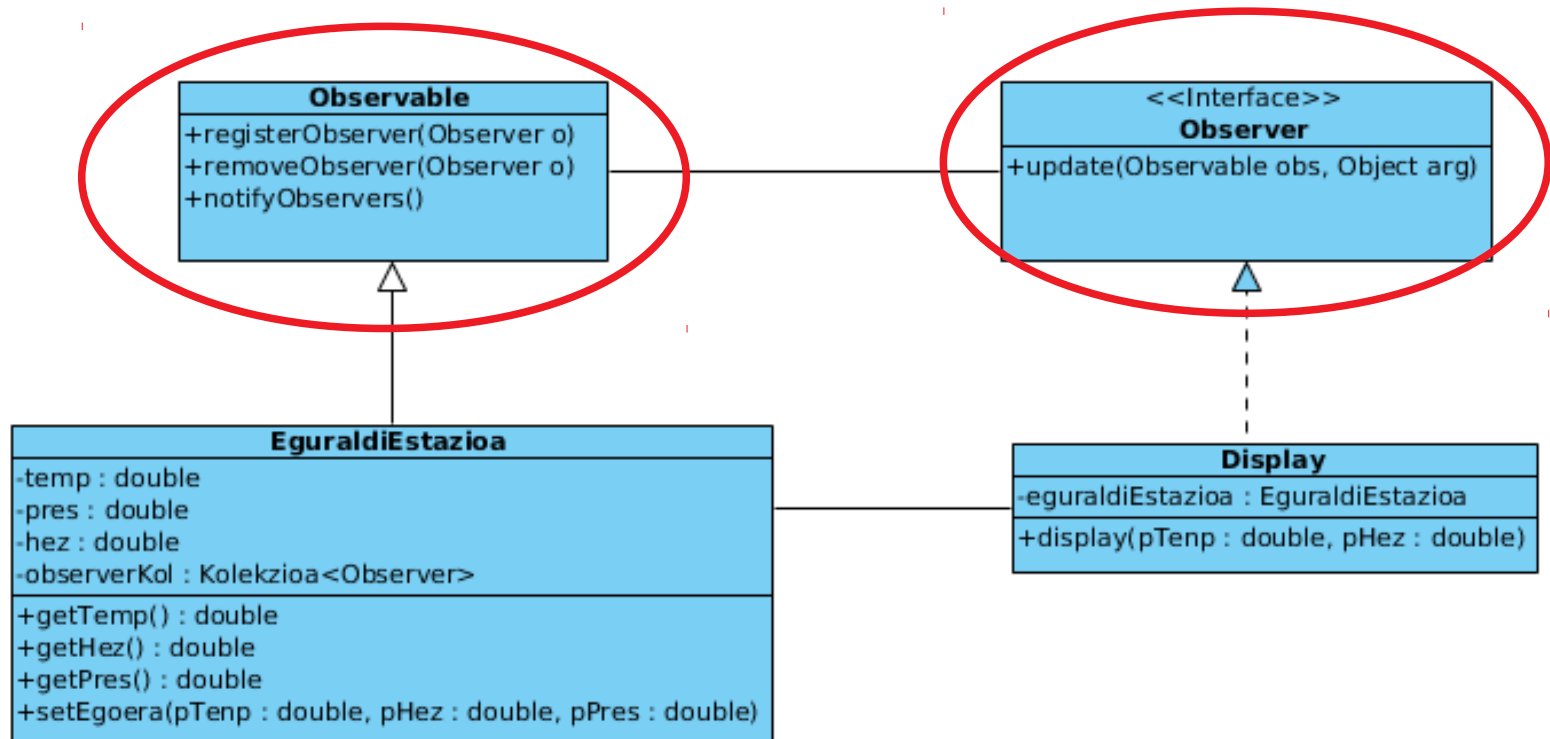
```
public class ProbaObserver {  
    public static void main(String[] args) {  
        EguraldiEstazioa eguEst = new EguraldiEstazioa();  
        Display display = new Display(eguEst);  
        eguEst.setEgoera(31.0, 0.42, 3.6);  
        eguEst.setEgoera(39.0, 3.42, 13.6);  
        eguEst.setEgoera(15.0, 7.42, 6.6);  
    }  
}
```



Eguraldi estazioan aldaketaren bat egon da!!
Momentuko datuak: 31.0 gradu eta 0.42% hezetasuna
Eguraldi estazioan aldaketaren bat egon da!!
Momentuko datuak: 39.0 gradu eta 3.42% hezetasuna
Eguraldi estazioan aldaketaren bat egon da!!
Momentuko datuak: 15.0 gradu eta 7.42% hezetasuna

Ebazpena

JAVAk baditu *Observer* eta *Observable* liburutegiak implementatuta. Guk horiek erabili!!!



Ebazpena: Java

Javak implementatuta ditu liburutegiak

```
import java.util.Observable;
import java.util.Observer;
import java.util.ArrayList;
```

```
public class EguraldiEstazioa extends Observable{
    private ArrayList observerKol;
    private double tenp;
    private double hez;
    private double pres;

    public EguraldiEstazioa() { observerKol = new ArrayList<>();}

    public void setEgoera(double pTenp, double pHez, double pPres)
    { //Eguraldian aldaketak egon dira!
        this.tenp = pTenp; this.hez = pHez; this.pres = pPres;
        egoeraAldatuDa();
    }

    public void egoeraAldatuDa() {
        setChanged();
        notifyObservers();
    }

    public double getTenp() {return tenp;}
    public double getHez() {return hez;}
    public double getPres() {return hez;}
}
```

```
public void setChanged(){
    changed = true;
}
```

```
public void notifyObservers(Object arg){
    if (changed == true) {
        gordeta dauden observer guztiei{
            update(this,arg);
        }
        changed = false;
    }
}
```

Ebazpena

```
import java.util.Observable;
import java.util.Observer;

public class Display implements Observer{

    private Observable eguraldiEstazioa;

    public Display (Observable pEguraldiEstazioa) {
        eguraldiEstazioa = pEguraldiEstazioa;
        eguraldiEstazioa.addObserver(this);
    }

    public void update(Observable obs, Object arg){
        System.out.println("Eguraldi estazioan aldaketaren bat egon da!!");
        display(((EguraldiEstazioa)obs).getTenp(), ((EguraldiEstazioa)obs).getHez());
    }

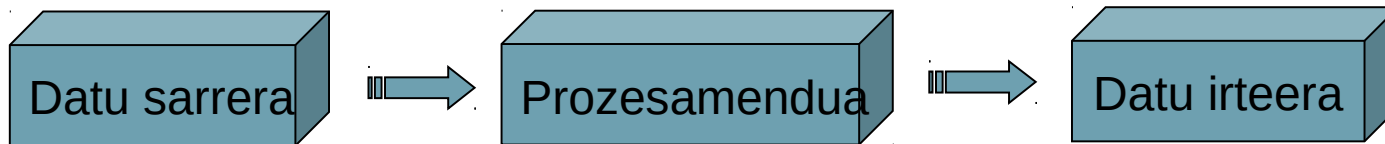
    public void display(double pTenp, double pHez) {
        System.out.println("Momentuko datuak: " + pTenp + "gradu eta " + pHez + "% hezetasuna");
    }

}
```


Model-View- Controller (MVC)

Model-View-Controller

Aplikazio guztietan hiru fase daude:

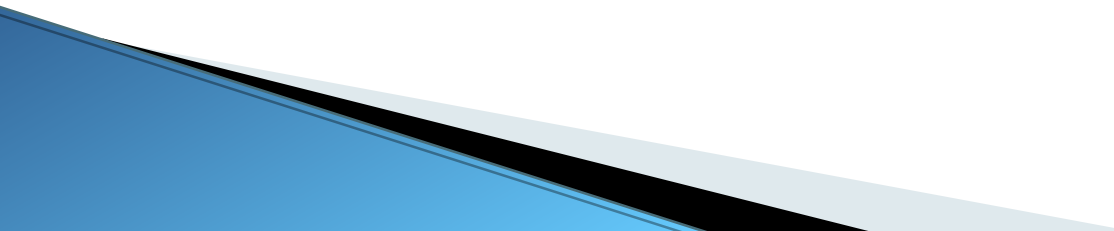


Azken horiei dagokion programazio modularra:

- ▶ Datu sarrera: Bista (GUI) + kontroladorea
- ▶ Prozesamendua : eredua
- ▶ Datu irteera (GUI)

Model-View-Controller

Aplikazio batetan, datuak, bista eta kontrol logika banatzen ditu:

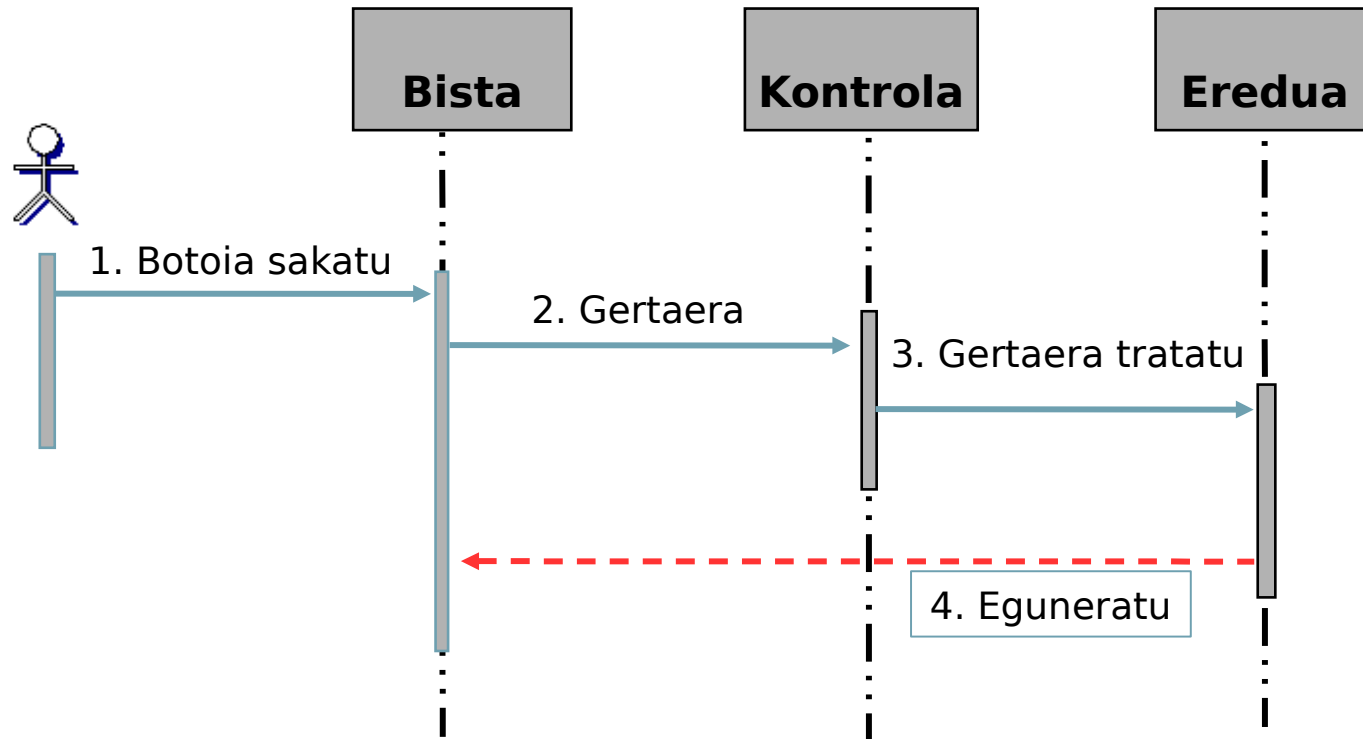
- ▶ **Eredua:** ebatzi beharreko arazoa.
 - ▶ **Bista:** eredu/erabiltzaile elkarrekintza ahalbidetzeko interfazea
 - ▶ **Kontroladorea:** erabakiak hartzen dituen kodea. Erabiltzailearen elkarrekintzei (gertaerei) erantzun, eta ereduan aldaketak eragiten ditu.
- 

Model-View-Controller

Hiru osagai horiek banatuta:

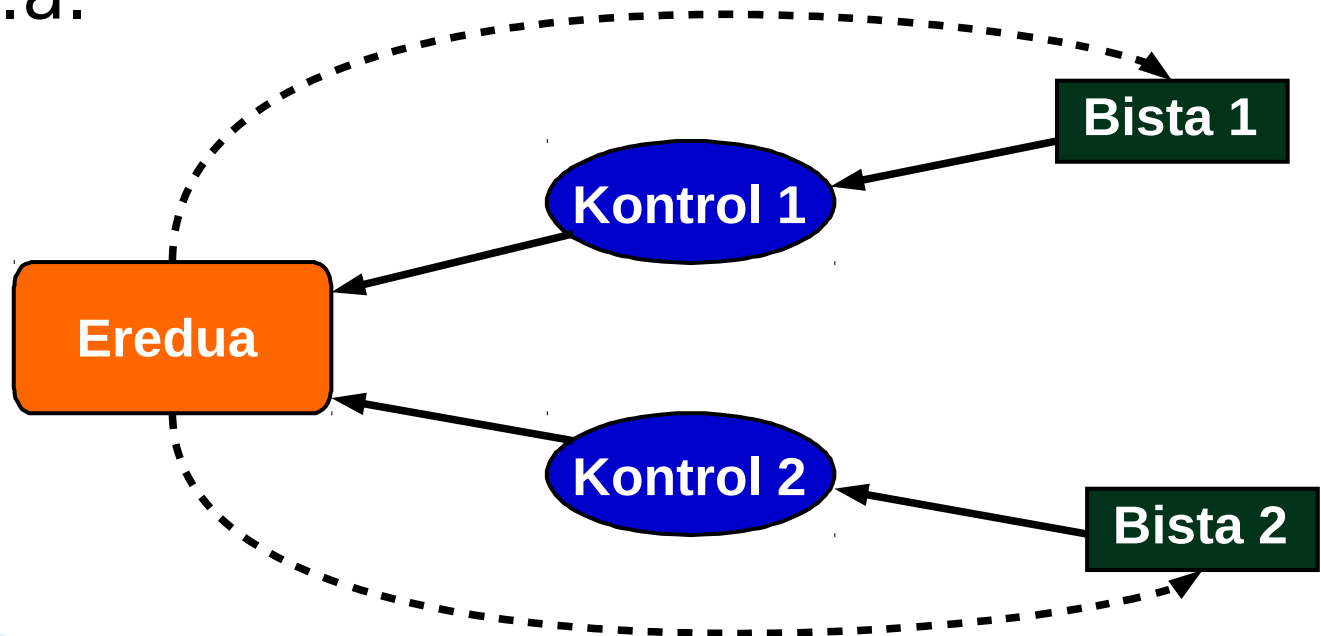
- ▶ Hiruretako edozein aldatzeko gai, besteen funtzionamendua ahalik eta gutxien ikututa.
- ▶ Berrerabilpena erraztu

MVC aplikazio baten funtzionamendua



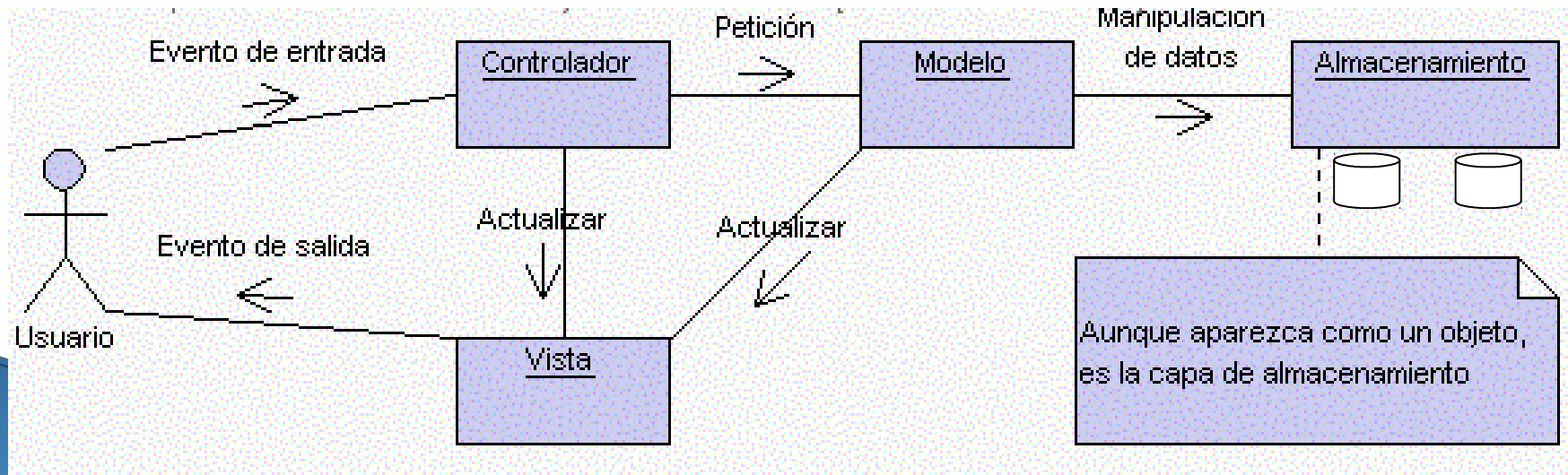
Model-View-Controller arteko menpekotasunak

Eredu batek bista ezberdinak izan ditzake. Adibidez, DB informazioa modu ezberdinetan aurkeztu daiteke: tarta diagrama, barrak, taulak, e.a.

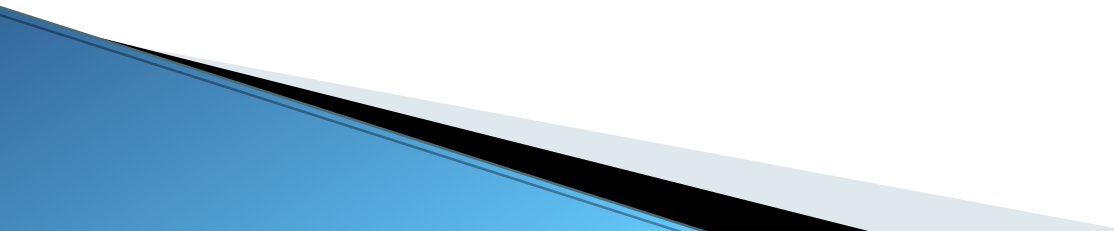


MVC aplikazio baten funtzionamendua

Aplikazio askok datu base bat erabiltzen dute datuak gordetzeko. MVC-ak ez du esplizituki datuak atzitzeko geruza hau aipatzen.



Ondorioak

- ▶ Osagai bakoitza independenteki garatu
 - ▶ Aldagarritasuna
 - ▶ Bista anitzak izateko aukera
 - ▶ Bistek ereduaren zatiak ikusi
 - ▶ Edozein aplikazio motara aplikagarria
- 

Erreferentziak

- ▶ Informazio gehiago:
 - Gamma, E. et al. *Designs Patterns, Elements of Reusable Object Oriented Software*. Addison Wesley.
 - Patterns Home Page: <http://hillside.net/patterns/>
 - Liburuak patroiei buruz: <http://hillside.net/patterns/books/>
 - <http://www.javacamp.org/designPattern/>
 - <http://www.dofactory.com/net/design-patterns>