Diseinu Patroiak

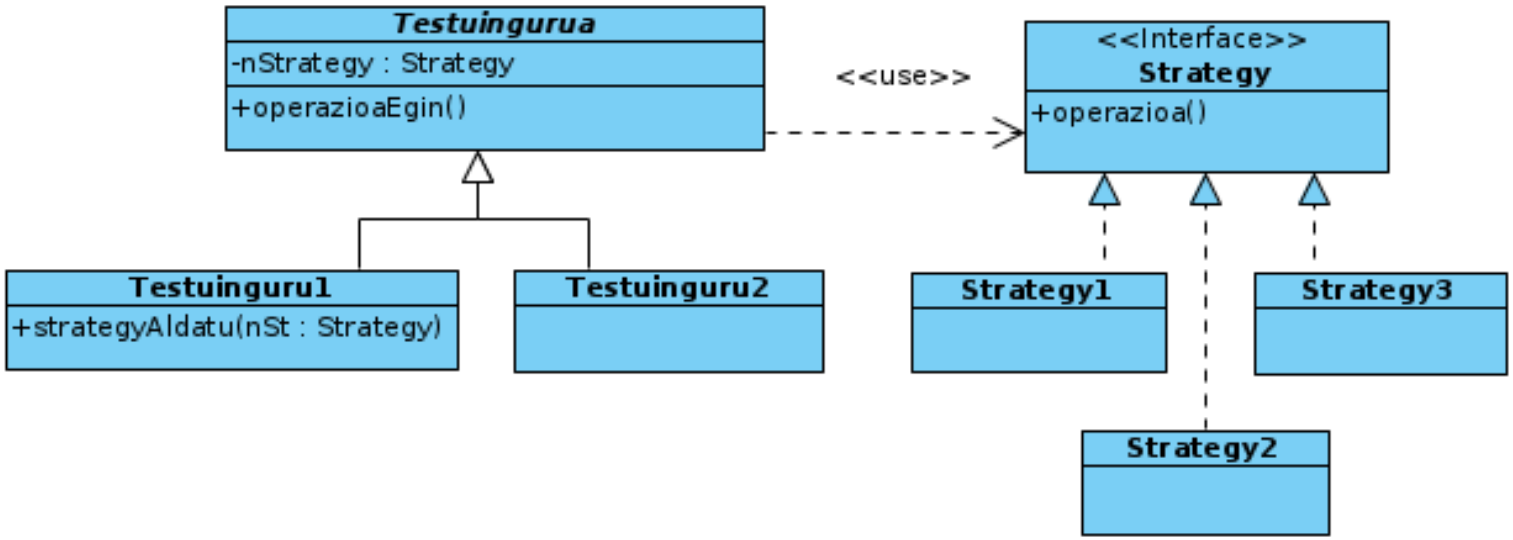
SOFTWARE INGENIARITZA

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.



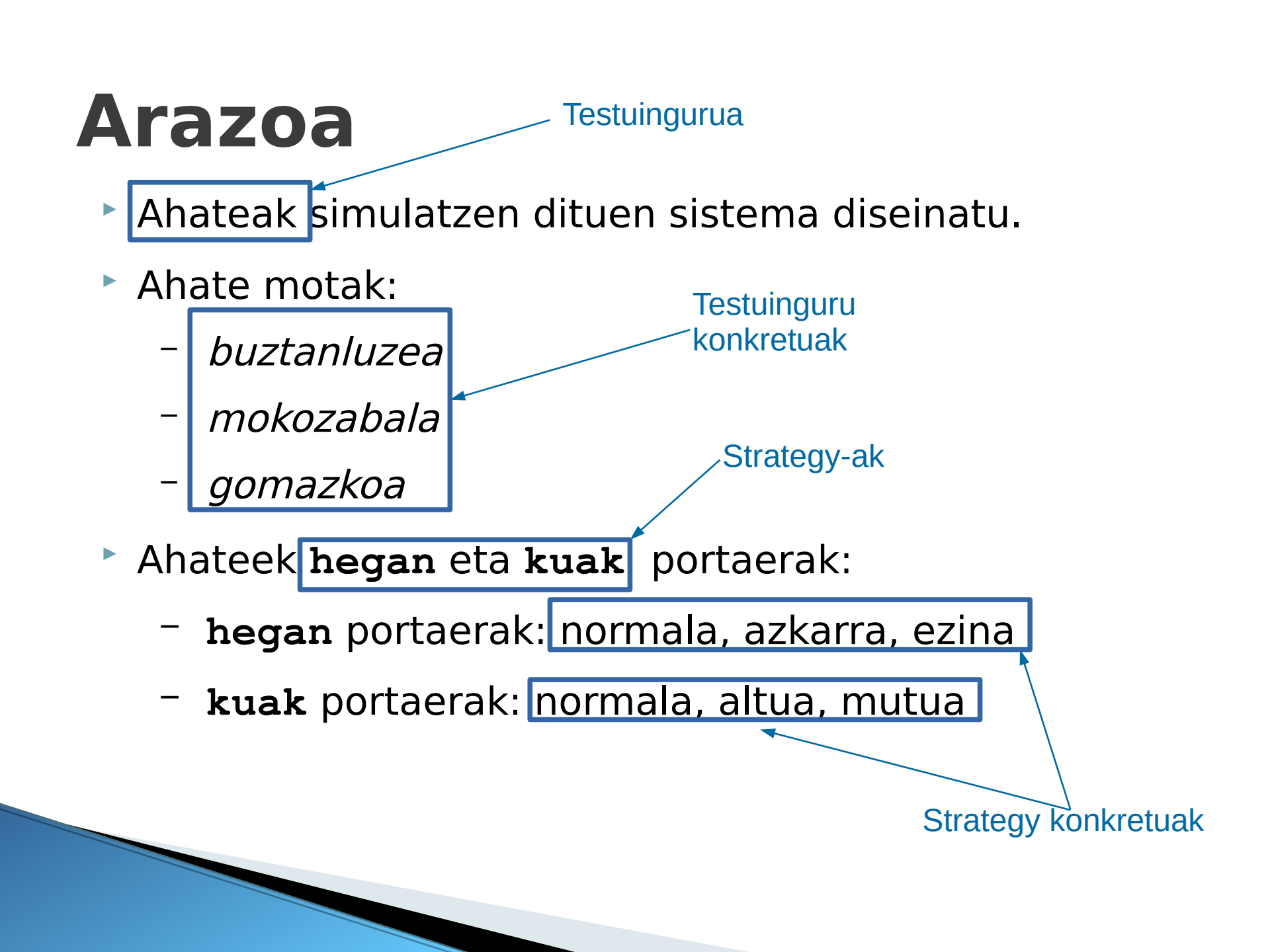
Ezaugarriak

- Testuingurua eta portaerak banatu
- Portaera aldagarriak, klase konkretuetan kapsulatu
- Berrerabilpena/hedapena hobetu
- Algoritmo familiak (estrategiak) definitu
- Bezeroak estrategia aldatu dezake

Arazoa

- Ahateak simulatzen dituen sistema diseinatu.
- Ahate motak:
  - *buztanluzea*
  - *mokozabala*
  - *gomazkoa*
- Ahateek **`hegan`** eta **`kuak`** portaerak:
  - **`hegan`** portaerak: normala, azkarra, ezina
  - **`kuak`** portaerak: normala, altua, mutua

Arazoa

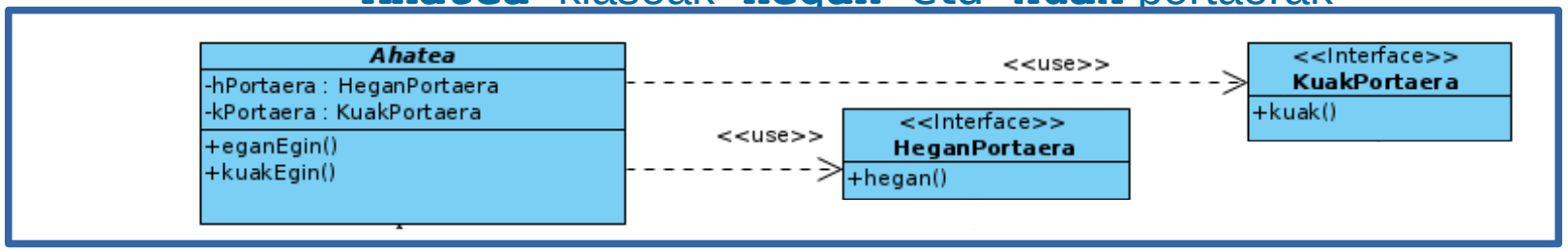

Testuingurua
Ahateak simulatzen dituen sistema diseinatu.
Ahate motak:
buztanluzea
mokozabala
gomazkoa
Testuinguru konkretuak
Strategy-ak
Ahateek hegan eta kuak portaerak:
hegan portaerak: normala, azkarra, ezina
kuak portaerak: normala, altua, mutua
Strategy konkretuak

Arazoa

- Ahate moten **`hegan`** eta **`kuak`** portaerak*:*
  - *Buztanluzea*: hegan normala eta kuak altua.
  - *Mokozabala*: hegan azkarra eta kuak normala. Gainera **`kuak`** portaera alda dezake.
  - *Gomazkoa*: hegan ezina eta kuak mutua.

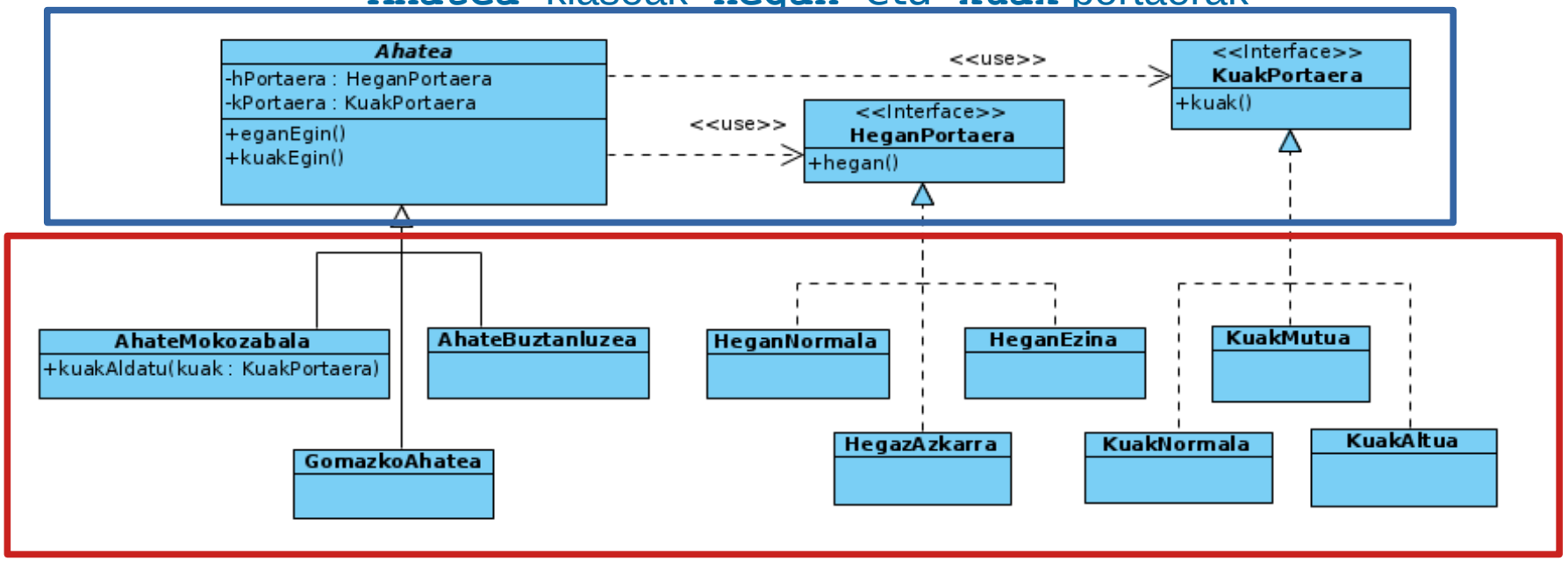
Ebazpena

`Ahatea` klaseak **`hegan`** eta **`kuak`** portaerak



Ebazpena

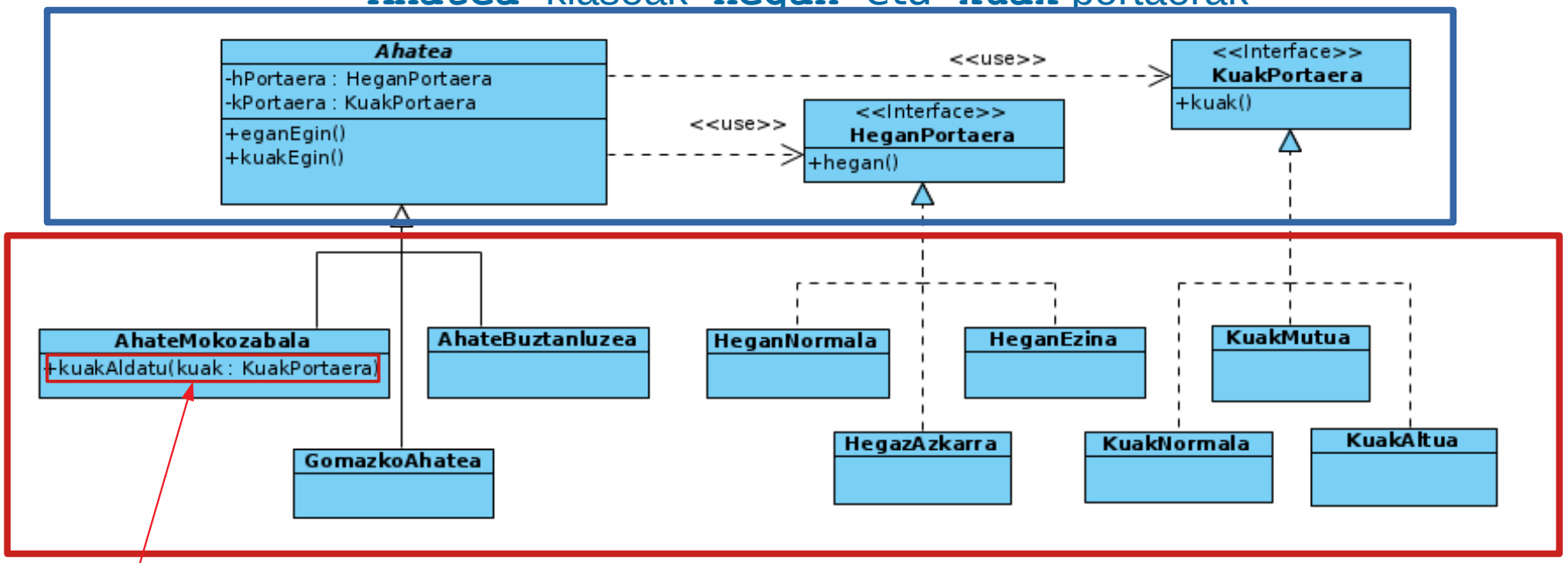
Ahatea klaseak **hegan** eta **kuak** portaerak



Ahatea-ren klase konkretuek **hegan** eta **kuak** portaera konkretuak

Ebazpena

Ahatea klaseak **hegan** eta **kuak** portaerak



Ahatea-ren klase konkretuek **hegan** eta **kuak** portaera konkretuak

AhateMokozabala klase konkretuak **kuak** portaera alda dezake

Ebazpena

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

public class HeganNormala implements HeganPortaera{
    public void hegan(){System.out.println("Normal hegan!");}
}

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

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

Ebazpena

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

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

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

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

Ebazpena

```
public abstract class Ahatea {

    protected HeganPortaera hPortaera;
    protected KuakPortaera kPortaera;

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

Portaerak

Portaeraren arabera, **`hegan`** eta **`kuak`** inplementazio desberdina

Ebazpena

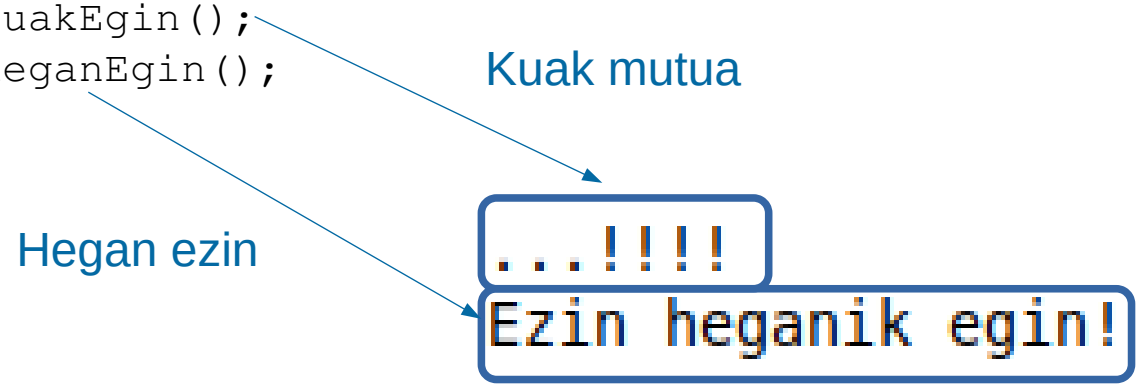
```
public class GomazkoAhatea extends Ahatea{
    public GomazkoAhatea(){
        hPortaera = new HeganEzina();
        kPortaera = new KuakMutua();
    }
}

public class AhateMokozabala extends Ahatea{
    public AhateMokozabala(){
        hPortaera = new HeganAzkarra();
        kPortaera = new KuakNormala();
    }
    public void kuakAldatu(KuakPortaera pKuakPortaera){
        kPortaera = pKuakPortaera;
    }

}
```

Ahate mota bakoitzaren portaerak eraikitzaileetan definitu

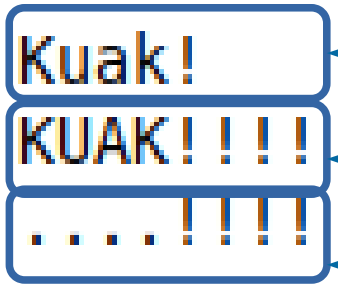
Ahate mokozabalak portaera aldatu dezake

Ebazpena

```
public class AhateSimuladorea {

    public static void main(String[] args) {

        Ahatea probaAhatea = new GomazkoAhatea();
        probaAhatea.kuakEgin();
        probaAhatea.heganEgin();
    }
}
```



Ebazpena



normala

altua

mutua

```
public class AhateSimuladorea {

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

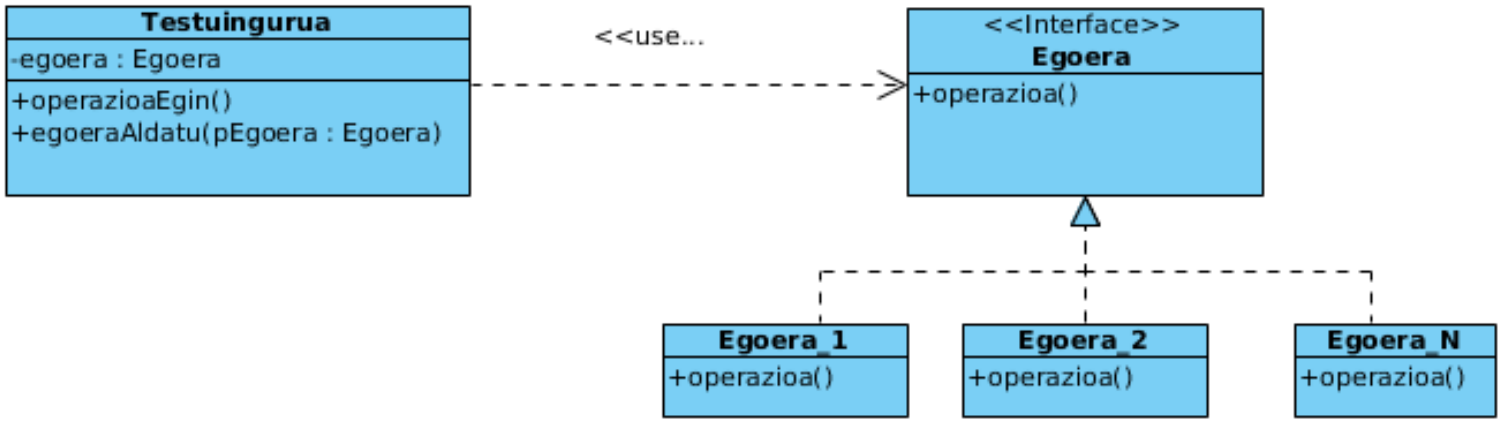
Ariketa: Audi Kotxeak

- Audi kotxeetako balazta kontrolatzeko sistema inplementatu.
- Bi balazta sistema daude, normala eta ABS.
- Hiru Audi modelo: *A1* (ABS), *A2* (normala) eta *A3* (ABS edo normala aukeratu)
- Sistemaren diseinua egin (klase diagrama)

State

Eskema Orokorra

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



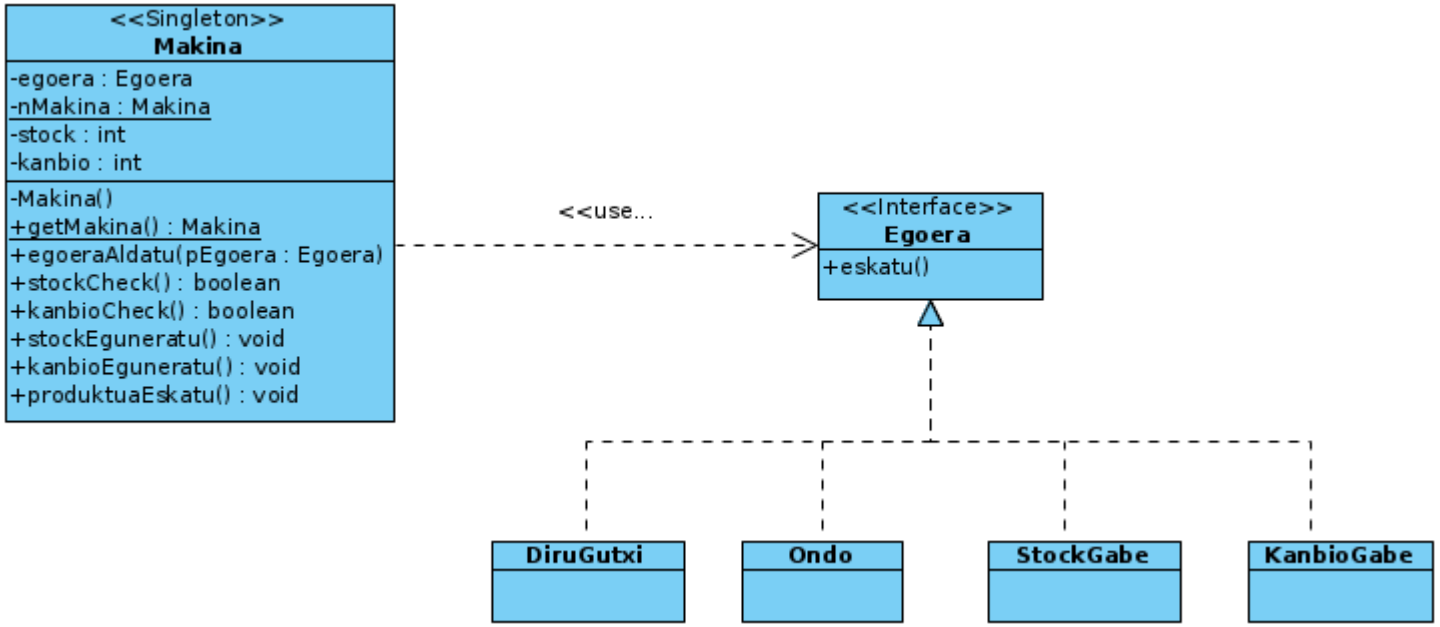
Ondorioak

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

Arazoa

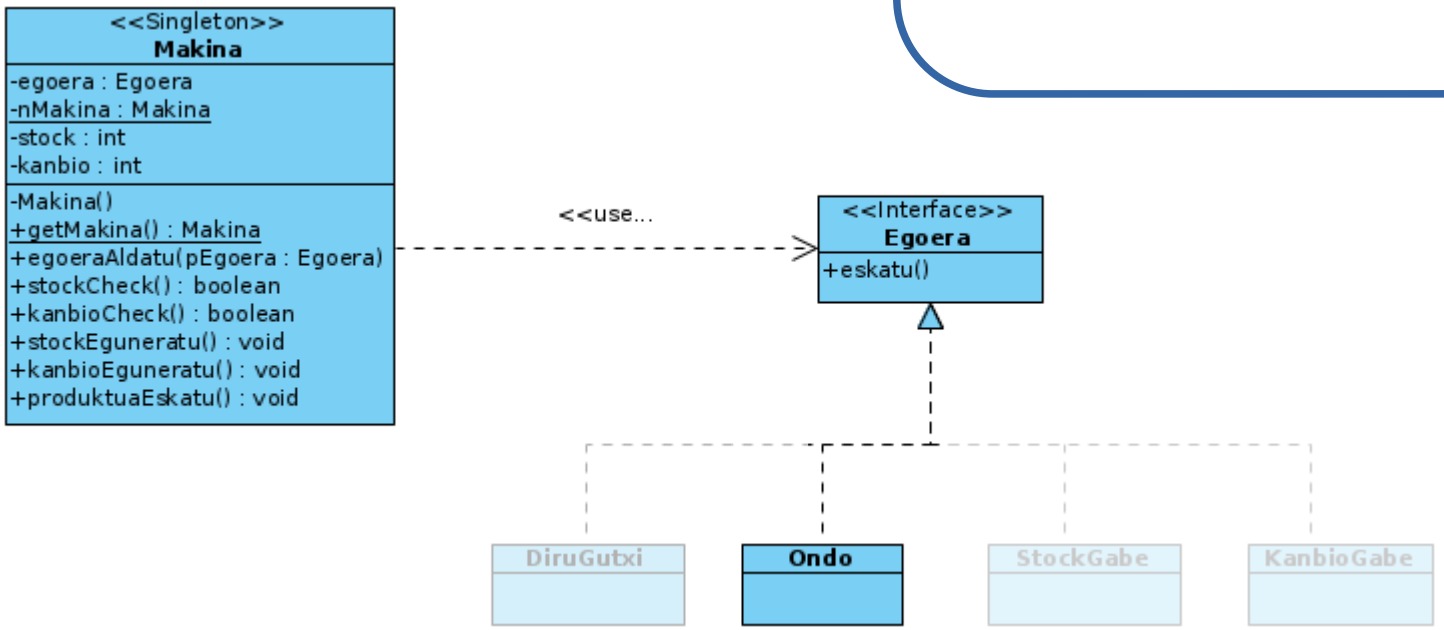
- Vending makinako **`eskaera`** botoia simulatu. Egoerak:
  - *Ondo* : diru nahikoa, stock-a eta kanbioak daude
    → produktua eman
  - *Stock gabe* : diru nahikoa, kanbioak daude, stock-ik ez
    → errore mezua
  - *Diru gutxi* : diru gutxiegi, stock-a eta kanbioak daude
    → diru gehiago eskatu
  - *Kanbiorik ez* : diru nahikoa, stock-a dago, kanbiorik ez
    → diru zehatza eskatu
- Funtzionalitatea diseinatu, egoera gehiago egon daitezkeela jakinda.

Ebazpena


<<Singleton>>
Makina
-egoera : Egoera
-nMakina : Makina
-stock : int
-kanbio : int
-Makina()
+getMakina() : Makina
+egoeraAldatu(pEgoera : Egoera)
+stockCheck() : boolean
+kanbioCheck() : boolean
+stockEguneratu() : void
+kanbioEguneratu() : void
+produktuaEskatu() : void
<<use...
<<Interface>>
Egoera
+eskatu()
DiruGutxi
Ondo
StockGabe
KanbioGabe

Ebazpena

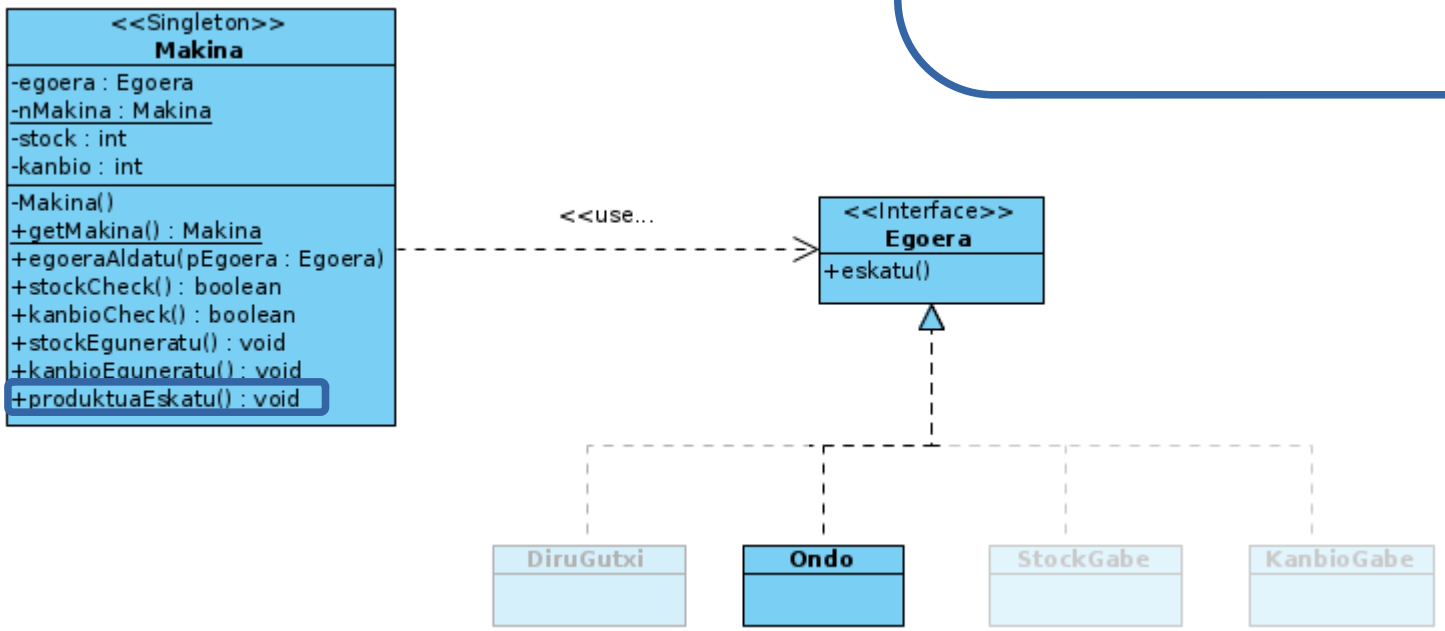
Ondo



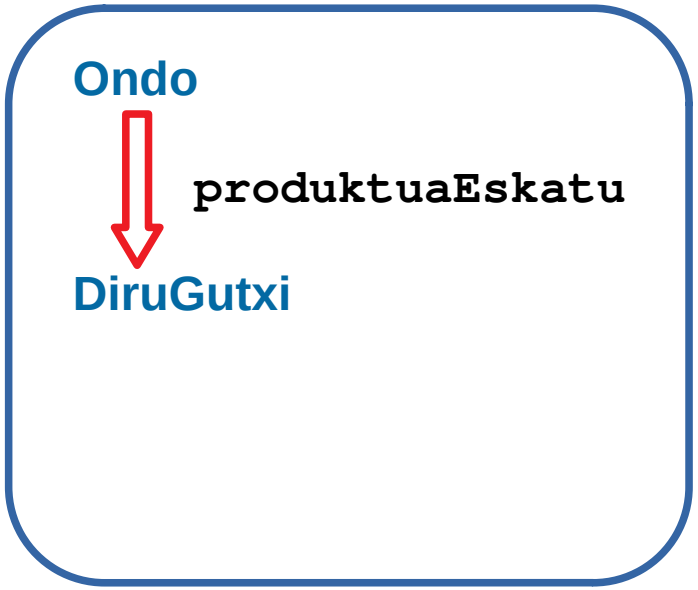
Ebazpena

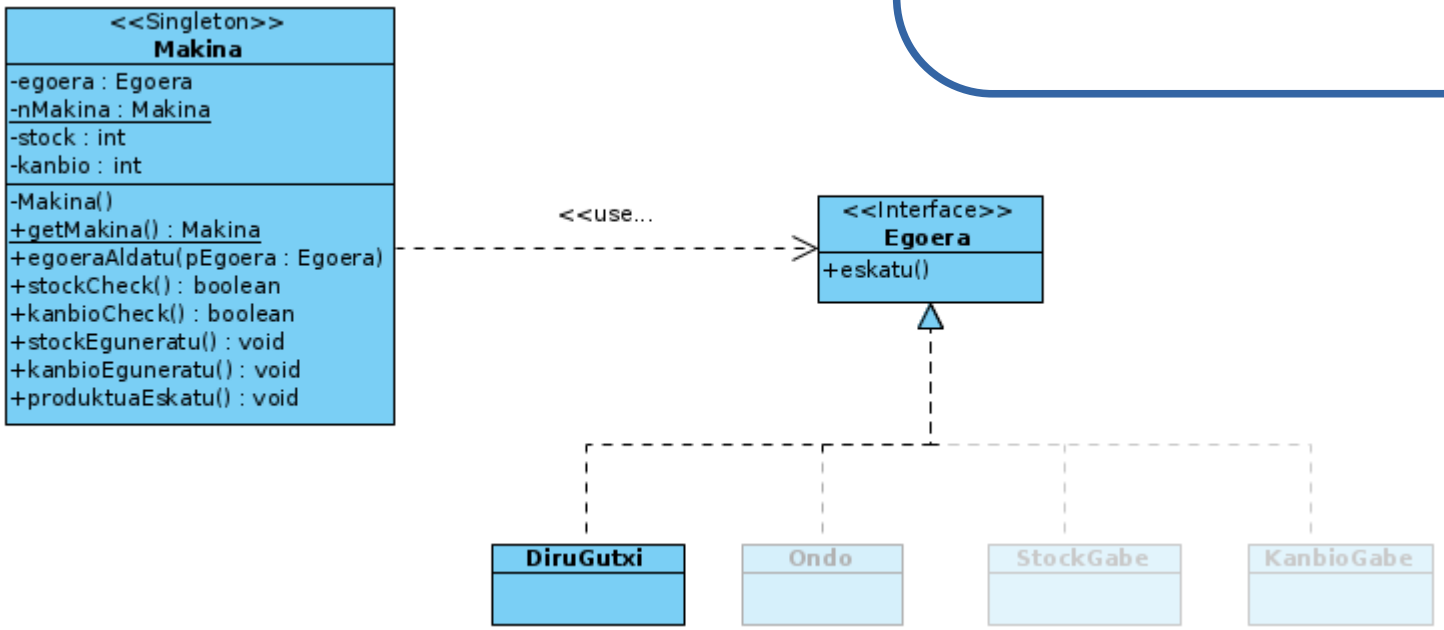
Ondo

`produktuaEskatu`

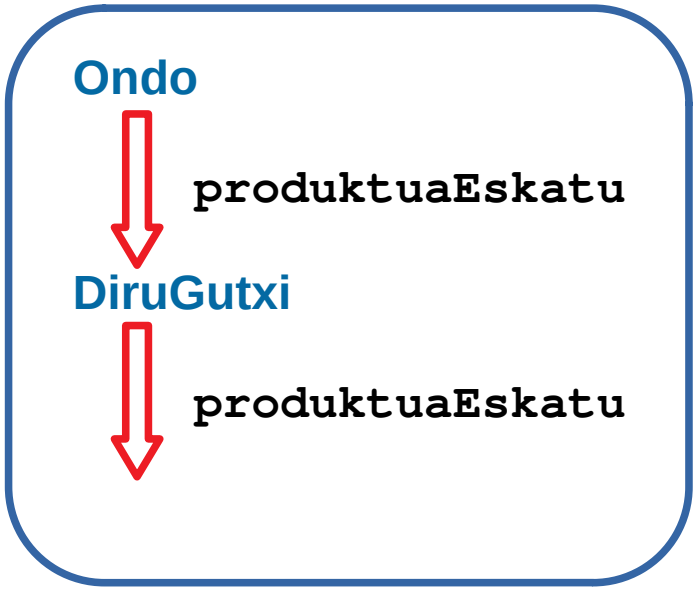


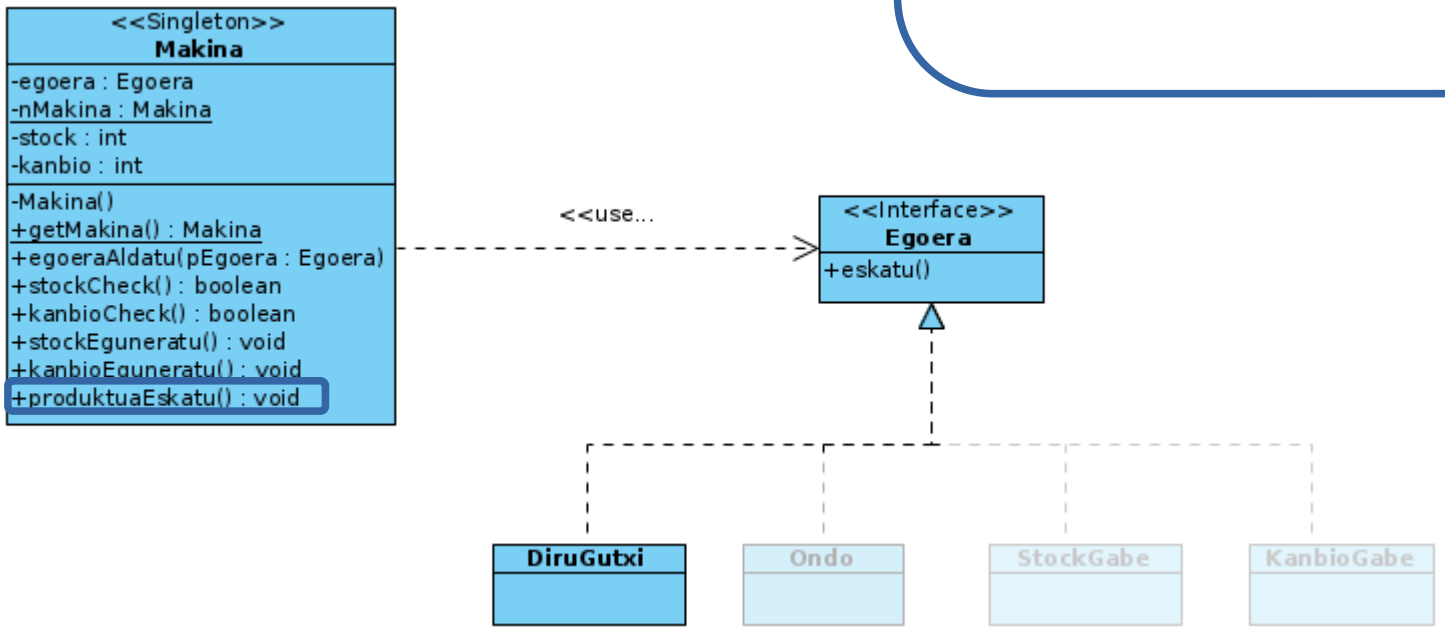
Ebazpena


Ondo
produktuaEskatu
DiruGutxi

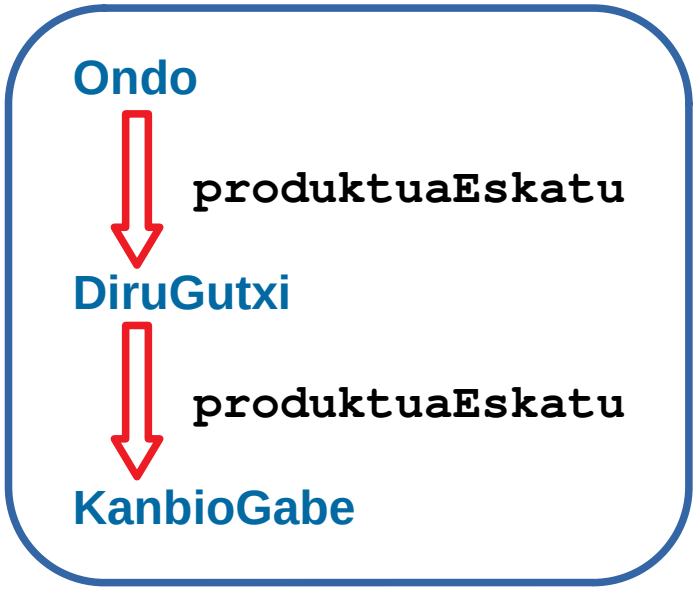

<<Singleton>>
Makina
-egoera : Egoera
-nMakina : Makina
-stock : int
-kanbio : int
-Makina()
+getMakina() : Makina
+egoeraAldatu(pEgoera : Egoera)
+stockCheck() : boolean
+kanbioCheck() : boolean
+stockEguneratu() : void
+kanbioEguneratu() : void
+produktuaEskatu() : void
<<use...
<<Interface>>
Egoera
+eskatu()
DiruGutxi
Ondo
StockGabe
KanbioGabe

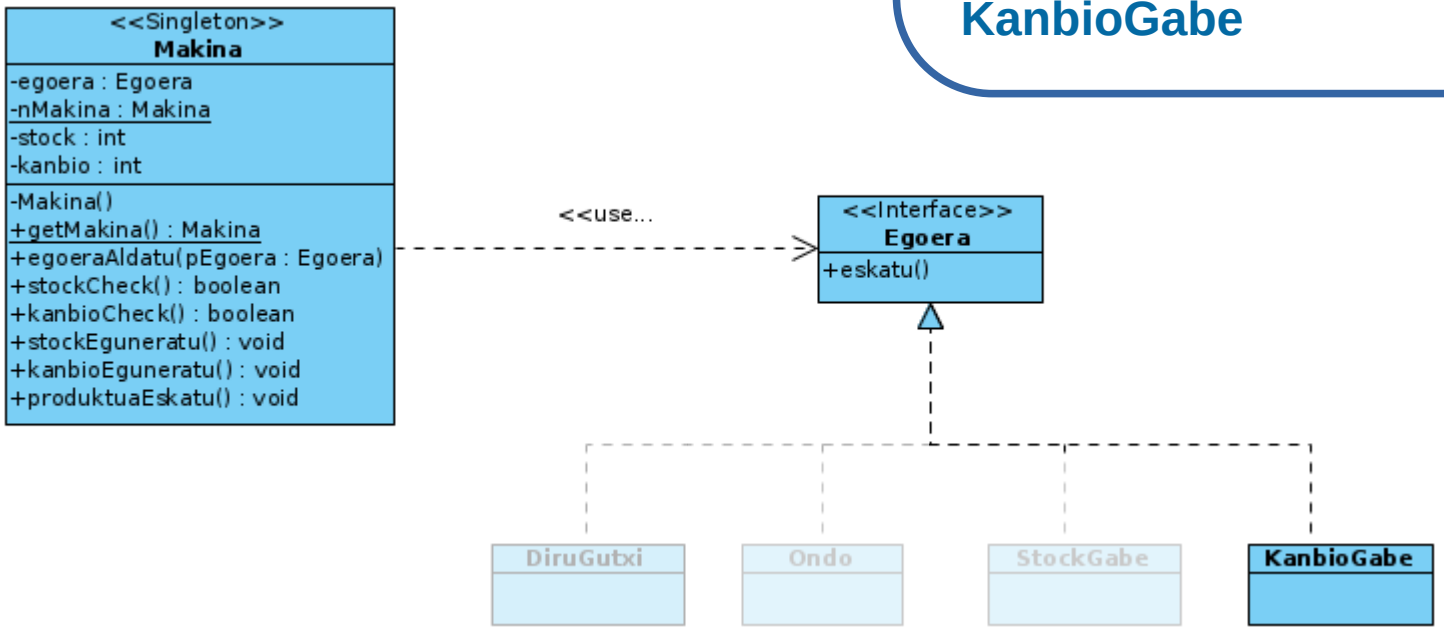
Ebazpena


Ondo
produktuaEskatu
DiruGutxi
produktuaEskatu


<<Singleton>>
Makina
-egoera : Egoera
-nMakina : Makina
-stock : int
-kanbio : int
-Makina()
+getMakina() : Makina
+egoeraAldatu(pEgoera : Egoera)
+stockCheck() : boolean
+kanbioCheck() : boolean
+stockEguneratu() : void
+kanbioEguneratu() : void
+produktuaEskatu() : void
<<use...
<<Interface>>
Egoera
+eskatu()
DiruGutxi
Ondo
StockGabe
KanbioGabe

Ebazpena


Ondo
produktuaEskatu
DiruGutxi
produktuaEskatu
KanbioGabe


<<Singleton>>
Makina
-egoera : Egoera
-nMakina : Makina
-stock : int
-kanbio : int
-Makina()
+getMakina() : Makina
+egoeraAldatu(pEgoera : Egoera)
+stockCheck() : boolean
+kanbioCheck() : boolean
+stockEguneratu() : void
+kanbioEguneratu() : void
+produktuaEskatu() : void
<<use...
<<Interface>>
Egoera
+eskatu()
DiruGutxi
Ondo
StockGabe
KanbioGabe

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(){ ... }

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

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

    public void stockEguneratu(){stock--;}
    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, egoeraz aldatuko da, eta portaera desberdina izango du

Makina klasean egoera aldaketarako metodoa, egoeretatik deituko da

Ebazpena

```
public interface Egoera {     public void eskatu(); }

public class Ondo implements Egoera{
     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 void eskatu(){
          System.out.println("-->Produktua ez dago stock-ean.");}
     }
}
```

Egoera barruan, egoera horri dagozkion **ekintzak** egiten dira

Egoera barruan, **egoera aldaketak** kudeatzen dira

Ebazpena

```
public class DiruGutxi implements Egoera{
    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 void eskatu(){
         System.out.println("--> Mesedez, diru zehatza sartu.");
        //Diru zehatza sartzen duenean
        System.out.println("--> Diru zehatza sartuta, produktua jasoko duzu.");
        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

    }
}
```

```
--> Produktua emango du.
--> Mesedez diru zehatza sartu.
--> Diru zehatza sartuta, produktua jasoko duzu.
--> Produktua ez dago stock-ean.
```

1. deia

2. deia

3. deia

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.

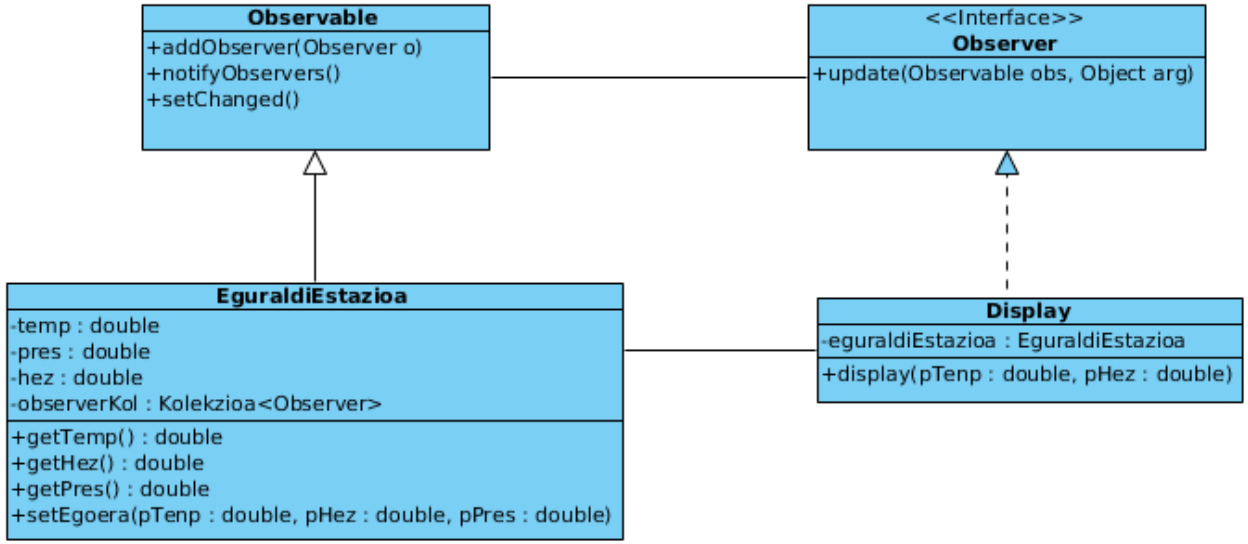
STATE vs. STRATEGY

- Eskema bera dute, zein da beraien arteko desberdintasuna?
  - Testuinguru/portaera
    - **`Strategy`**: hainbat testuinguru, bakoitzak bere portaera
    - **`State`**: testuinguru bakarra, hainbat egoerarekin
  - Erabiltzailearen ikuspuntua
    - **`Strategy`**: estrategiak ezagutu, baita aukeratu ere
    - **`State`**: ez daki ezer barne egoeren inguruan. Ezin 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.

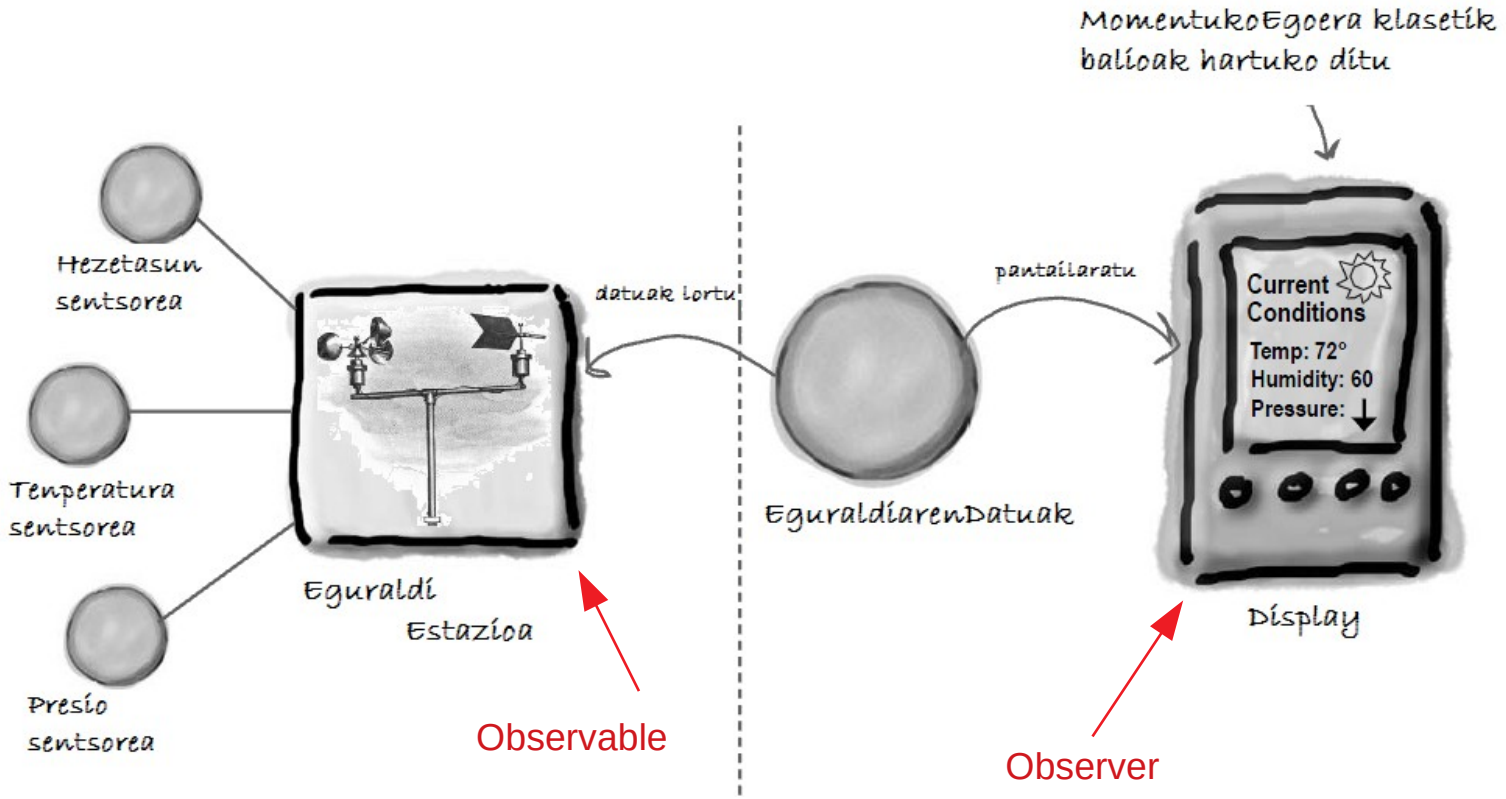


Ondorioak

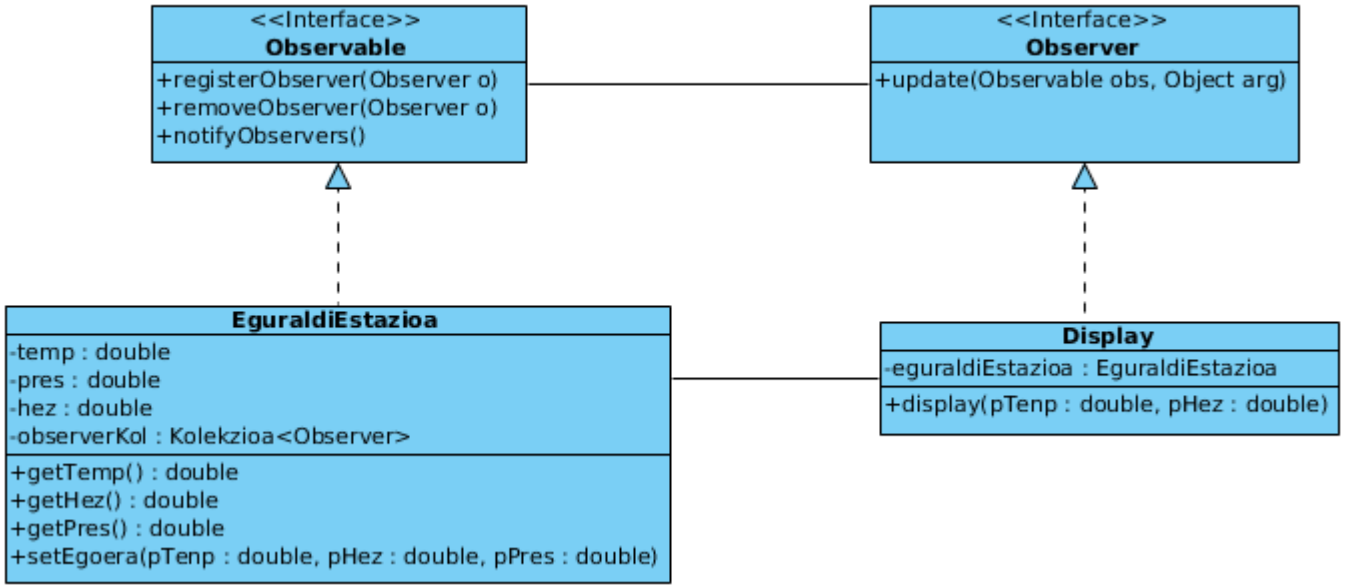
- Akoplamendu soltea (loose-coupling); *observable*-ak ez daki ezer*observer* motaren inguruan.
- Entzuleei jakinarazpenak bidali
- Hedagarria

Arazoa

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



Ebazpena


<<Interface>>
Observable
+registerObserver(Observer o)
+removeObserver(Observer o)
+notifyObservers()
<<Interface>>
Observer
+update(Observable obs, Object arg)
EguraldiEstazioa
-temp : double
-pres : double
-hez : double
-observerKol : Kolekzioa<Observer>
+getTemp() : double
+getHez() : double
+getPres() : double
+setEgoera(pTenp : double, pHez : double, pPres : double)
Display
-eguraldiEstazioa : EguraldiEstazioa
+display(pTenp : double, pHez : double)

Ebazpena: Java

```
public class EguraldiEstazioa implements Observable{

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

     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);
          }
     }

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

Observer kolekzioa

Observable-n egoera aldaketa simulatzeko

Ebazpena: Java

Egoera aldaketa detektatzerakoan **`Observer`-ei jakinarazpena**

```
    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;}
}
```

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");
     }

}
```

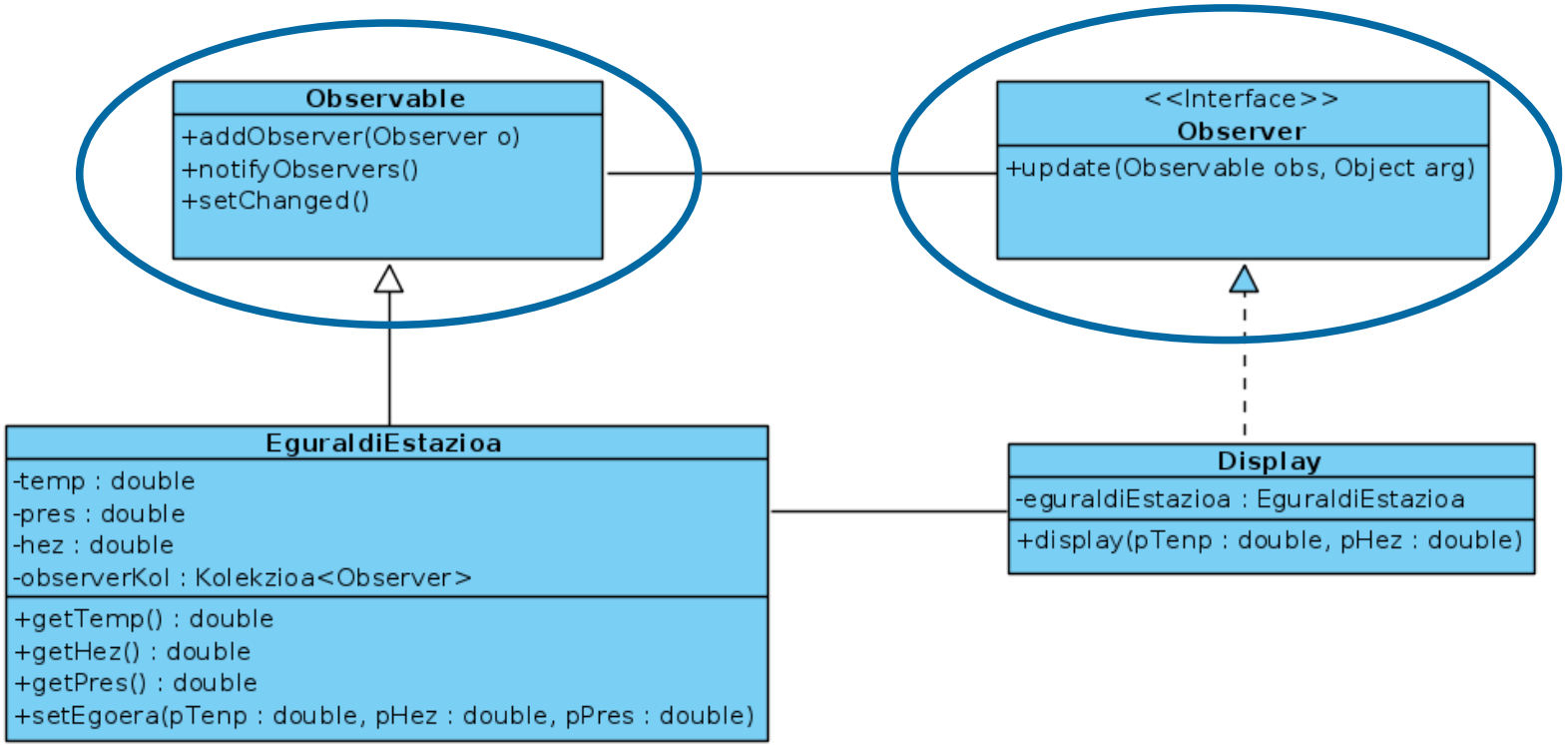
Ebazpena: Java

```
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* inplementatuta. Guk horiek erabili!!!



Ebazpena: Java

Javak inplementatuta ditu liburutegiak

```
import java.util.Observable;


public class EguraldiEstazioa extends Observable{

     private double tenp;
     private double hez;
     private double pres;

     public EguraldiEstazioa() {}

     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;
     }
}
```

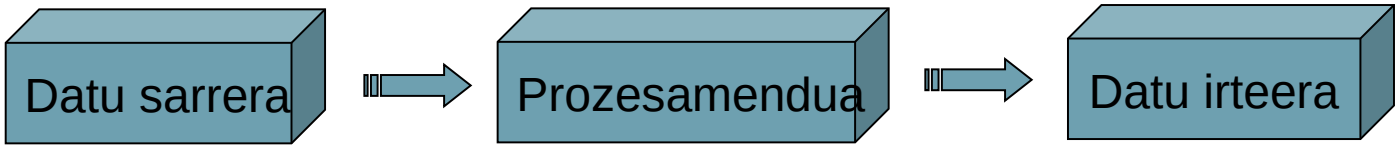
Ebazpena

```
import java.util.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:

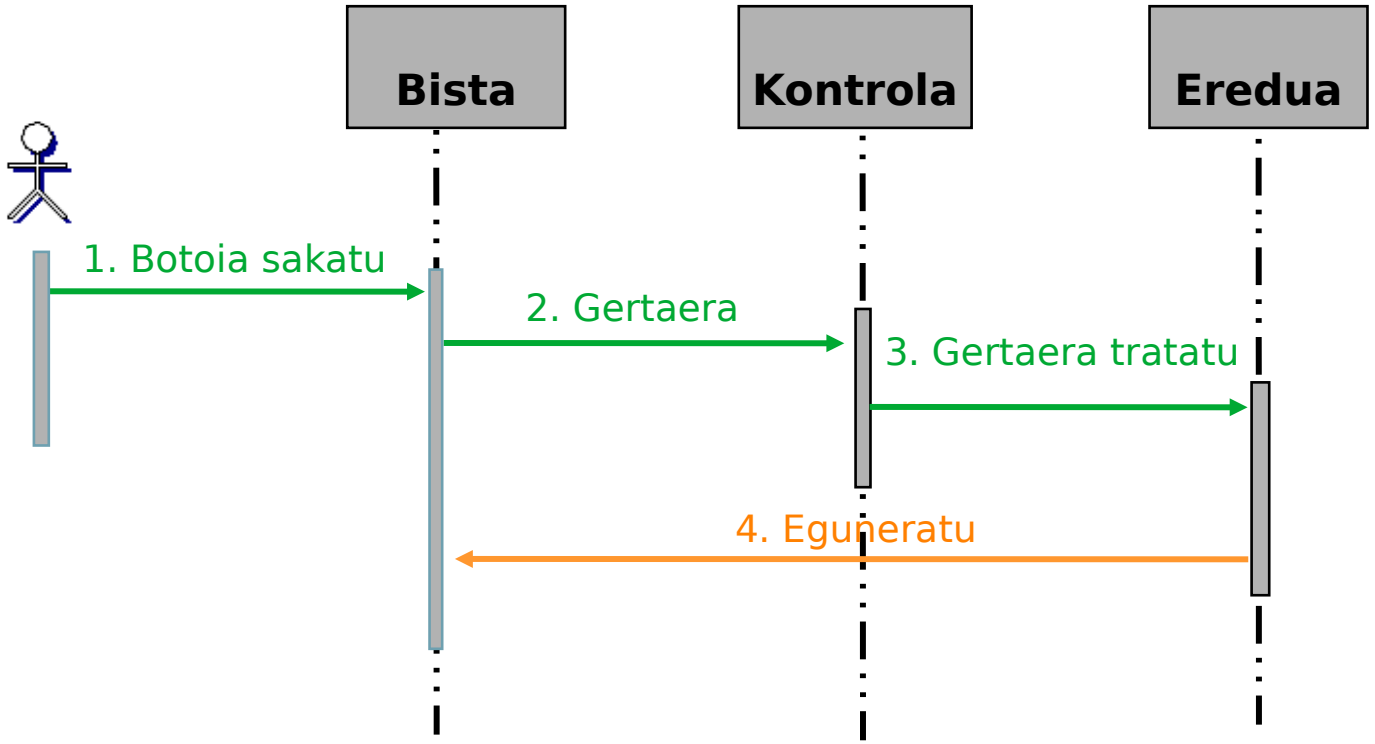
- **Eredua:** ebatzi beharreko arazoa.
- **Bista:** eredu/erabiltzaile elkarrekintza ahalbidetzeko interfazea
- **Kontroladorea:** erabakiak hartzen dituen kodea. Erabiltzailearen elkarrekintzei (gertaerei) erantzun, eta ereduan aldaketak eragiten ditu.

Model-View-Controller

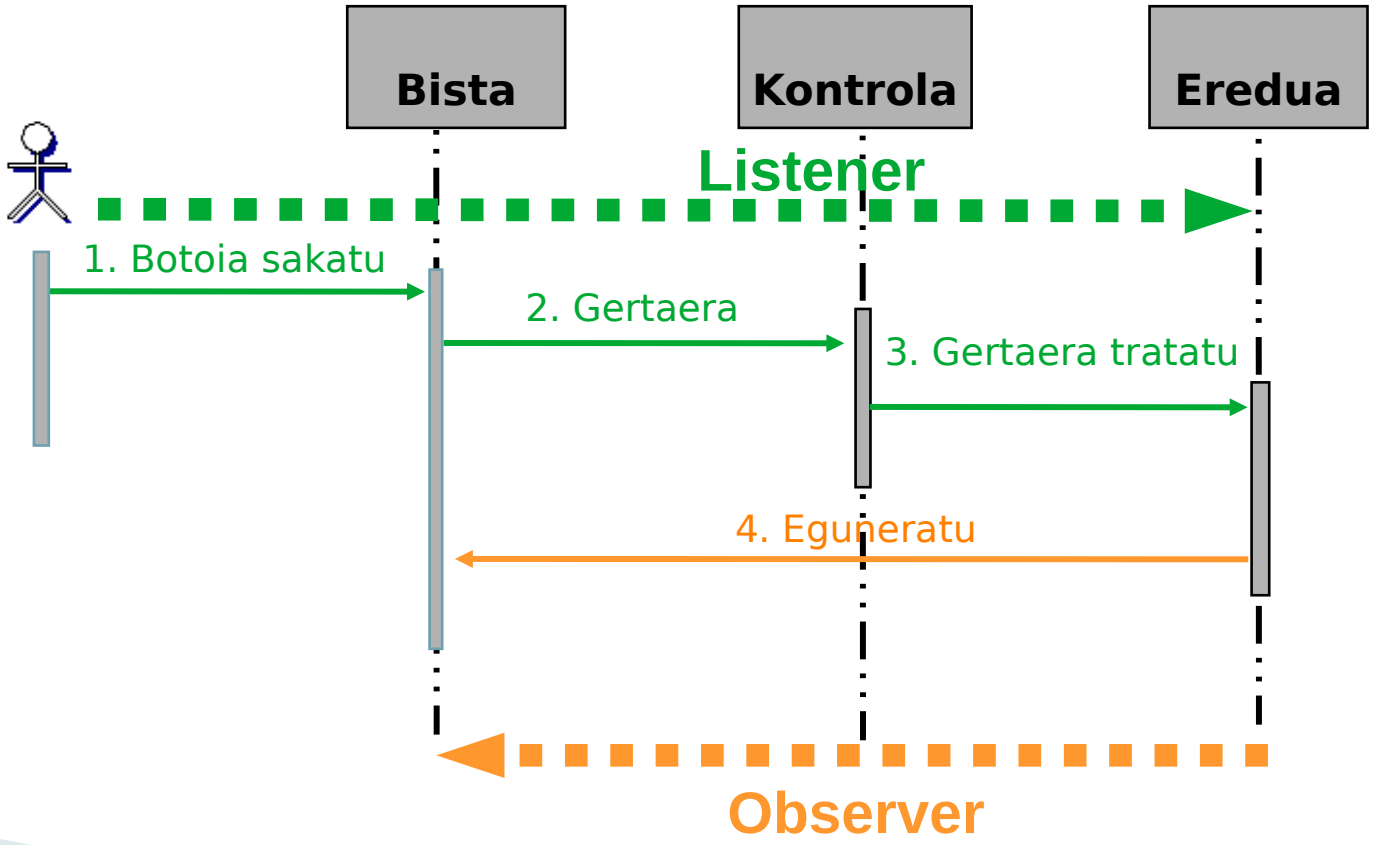
Hiru osagai horiek banatuta:

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

MVC aplikazio baten funtzionamendua



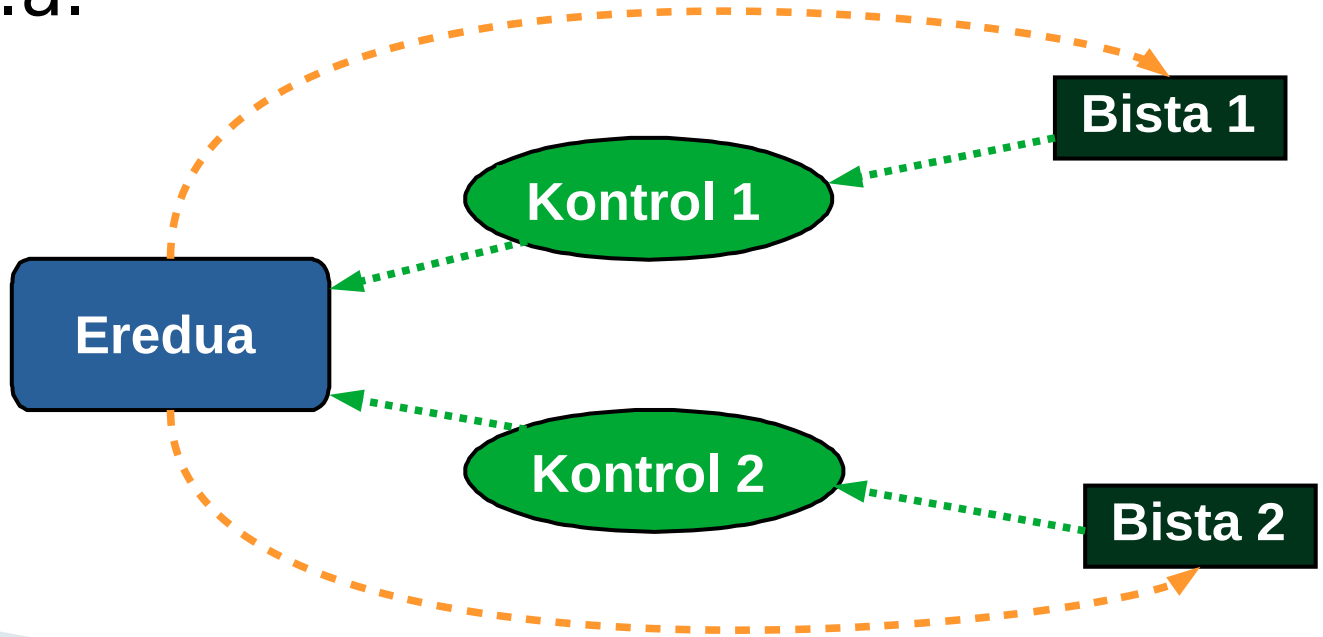
MVC aplikazio baten funtzionamendua



Model-View-Controller arteko menpekotasunak

Eredu batek bista ezberdinak izan ditzake.

Adibidez, DB informazioa modu ezberdinetan aurkeztu daiteke: tarta diagrama, barrak, taulak, e.a.



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