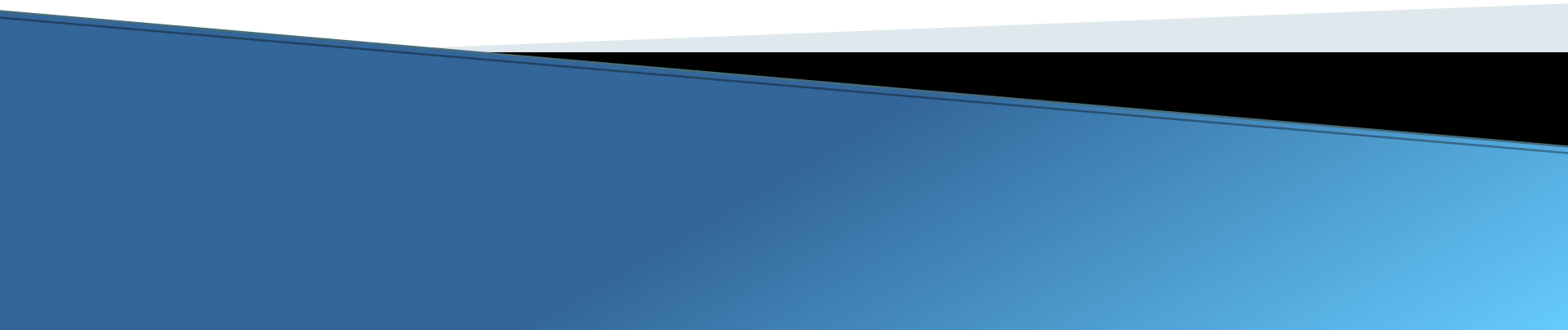


Diseinu Patroiak

SOFTWARE INGENIARITZA



Portaerazkoak

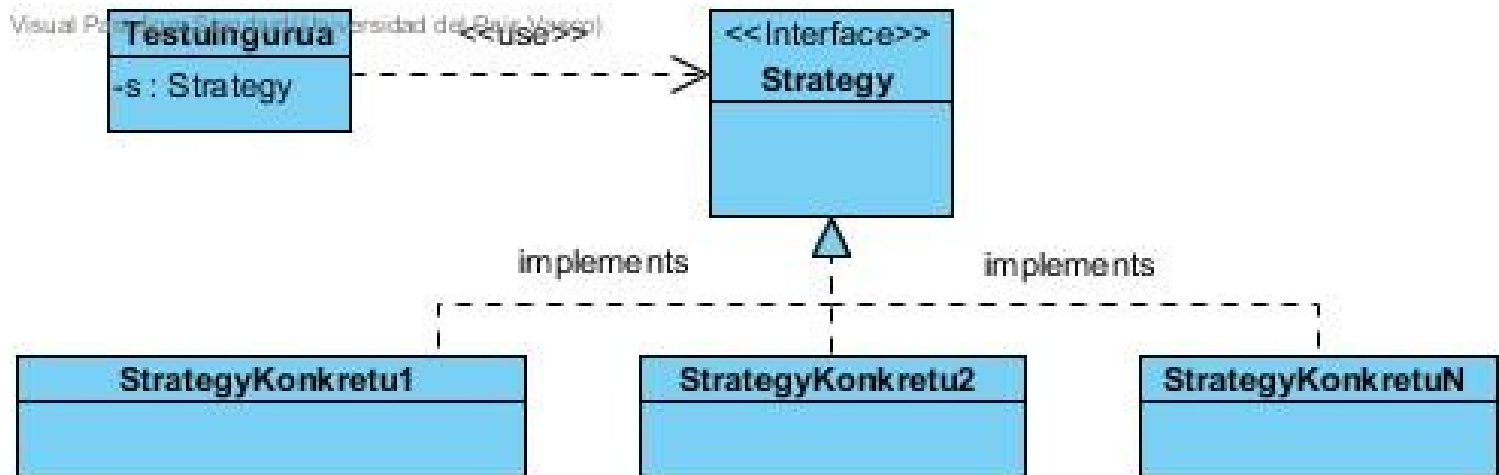


Strategy




Eskema Orokorra

Strategy: algoritmo familia bat definitzen du, horietako bakoitza enkapsulatzen, eta beren artean trukagarri egiten ditu. Gainera, algoritmoa “run-time”an aldatzea ahalbidetzen du



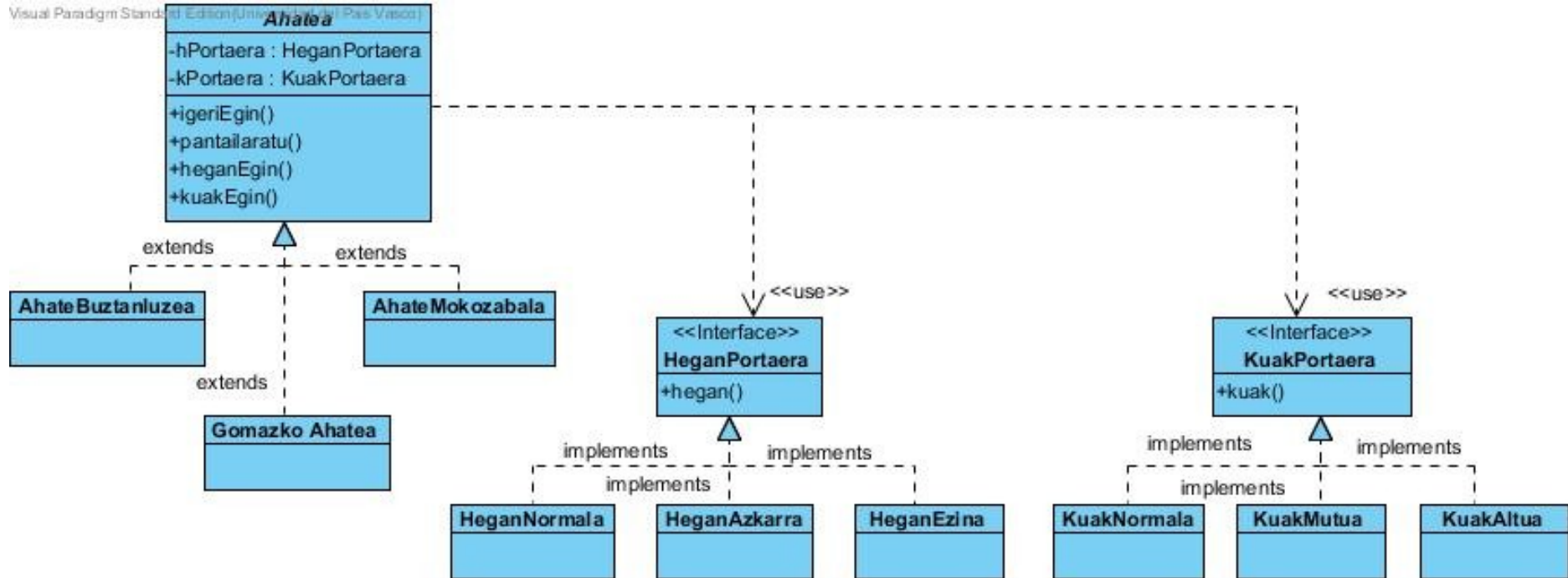
Ondorioak

- ▶ Testuingurua eta portaerak banatu
 - ▶ Berrerabilpena hobetu
 - ▶ Algoritmo familiak definitu
 - ▶ Inplementazio aukera desberdinak
 - ▶ Bezeroak strategy desberdinak aukeratzen ditu
- 

Arazoa

- ▶ Ahateak simulatzen dituen sistema diseinatu
- ▶ Ahateek portaera desberdinak: *hegan* eta *kuak* egin, besteak beste.
 - Hegan eta kuak egiteko era desberdinak
- ▶ Sistemaren klase diagrama egin, ahate mota eta portaera berriak gehitu daitezkeela kontutan hartuz.

Ebazpena



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 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!");}  
}
```

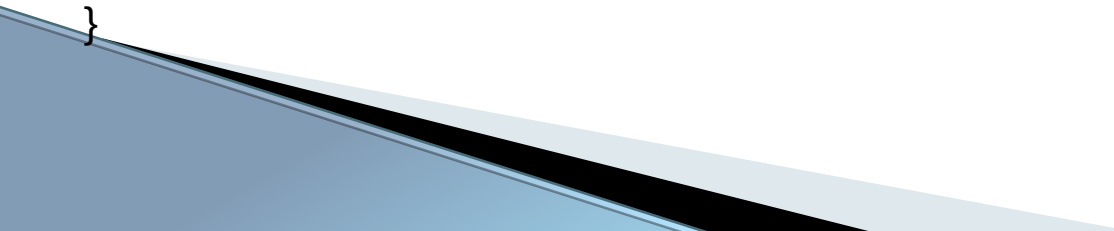
Ebazpena

```
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!");}  
}
```

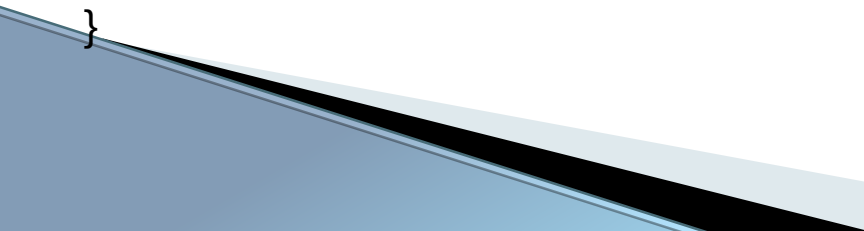
```
public class KuakMutua implements KuakPortaera{  
    public KuakMutua(){  
    public void kuak(){System.out.println("...!");}  
}
```



Ebazpena

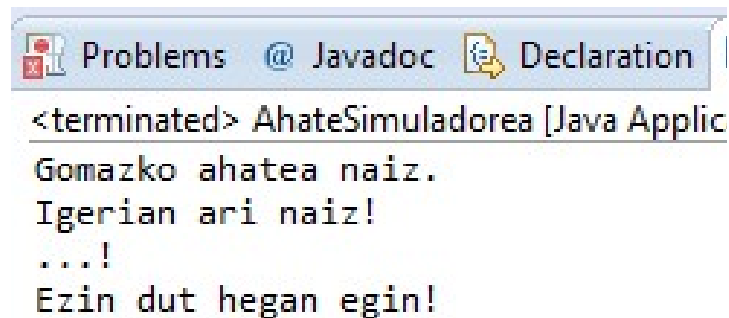
```
public class GomazkoAhatea extends Ahatea{
    public GomazkoAhatea(){
        hPortaera = new HeganEzina();
        kPortaera = new KuakMutua();
    }
    public void pantailaratu(){System.out.println("Gomazko ahatea naiz.");}
}

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



Ebazpena

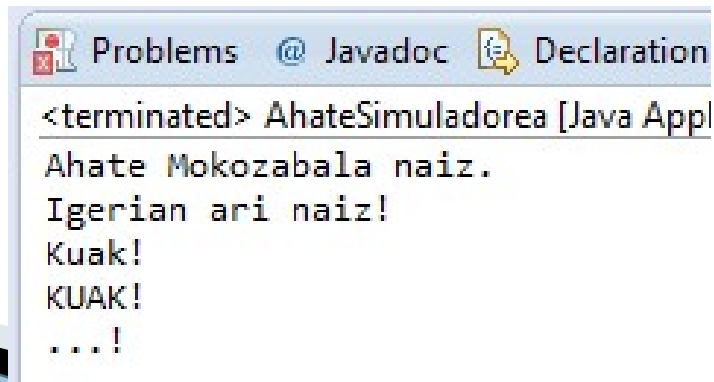
```
public class AhateSimuladorea {  
  
    public static void main(String[] args) {  
  
        Ahatea probaAhatea = new GomazkoAhatea();  
        probaAhatea.pantailaratu();  
        probaAhatea.igeriEgin();  
        probaAhatea.kuakEgin();  
        probaAhatea.heganEgin();  
    }  
}
```



Ebazpena

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

Dinamikoki ari
naiz portaera
aldatzen



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

Ariketa: Audi Kotxeak

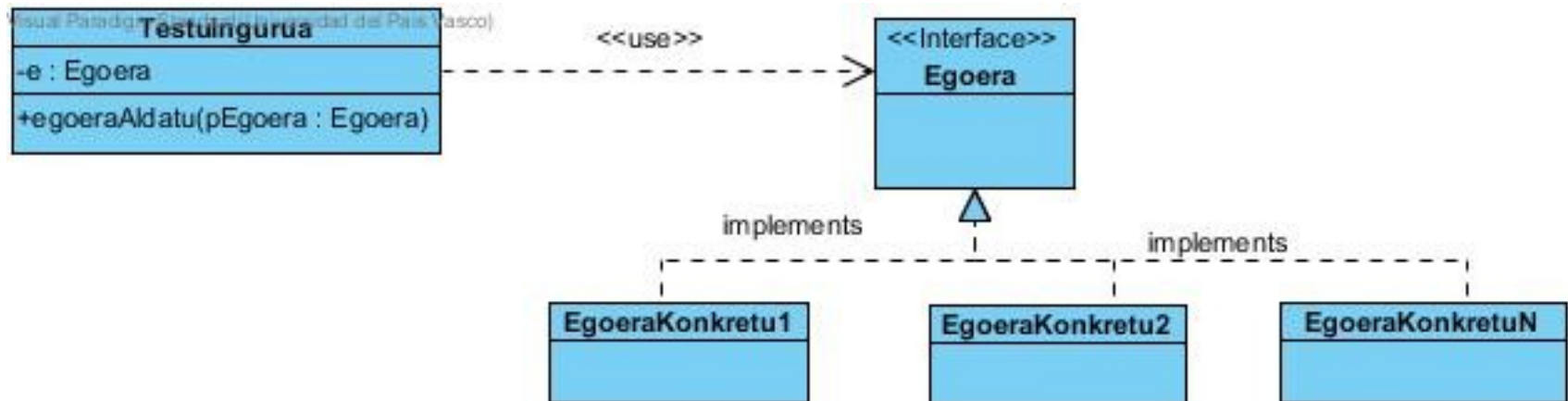
- ▶ Audi kotxeak kontrolatzen dituen sisteman lanean jarraitzen dugu.
- ▶ Balazta zapaltzean gertatzen dena kontrolatzeko sistema inplementatu.
- ▶ Bi balazta sistema daude, normala eta ABS.
- ▶ Sistemaren diseinua egin (klase diagrama), Audiko kotxe eredu gehienek bi balazta sistema horietako bat dutela jakinda.
- ▶ Zer egin beharko litzateke kotxe baten balazta sistema aldatzeko?

State

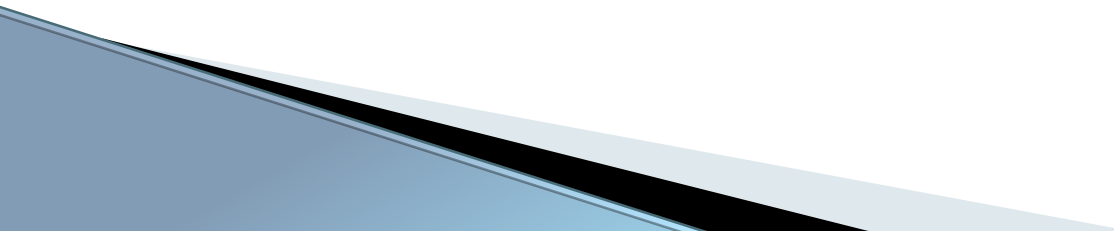
The image features a minimalist design with a white background. The word "State" is written in a bold, dark gray sans-serif font on the right side. A thin horizontal line is positioned below the text. The bottom of the image is decorated with a blue gradient that transitions from a dark blue on the left to a lighter blue on the right, with a solid black horizontal band running through the middle of the gradient.

Eskema Orokorra


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



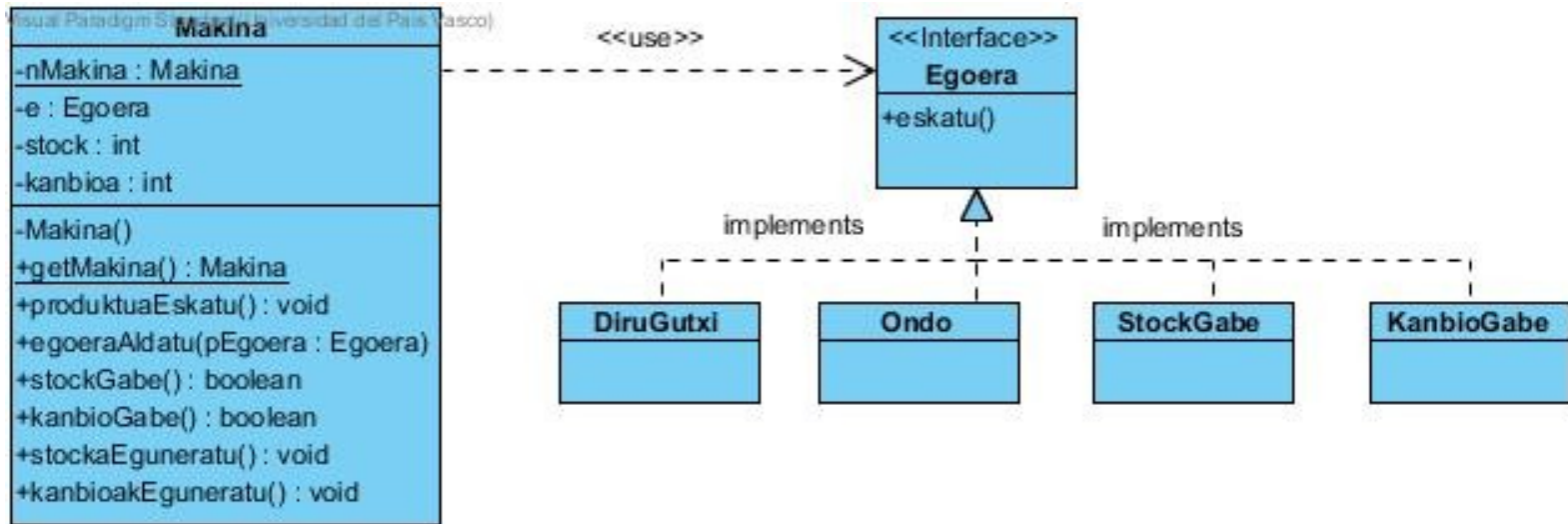
Ondorioak

- ▶ Egoeraren arabera, portaera desberdina
 - ▶ Egoera bakoitzeko portaera kapsulatu
 - ▶ Egoeren arteko transizio esplizituak
 - ▶ Hedagarria
 - ▶ Bezeroak egoeren inguruko informazio gutxi edo ezer
- 

Arazoa

- ▶ Vending makina bat kontrolatzeko sistema. Eskaera botoiari zaplatzean duen portaera diseinatuko dugu.
 - ▶ Diru nahikoa sartu bada eta stock-ik badago, produktua eman.
 - ▶ Diru nahikoa sartu bada, baina, stock-ik ez badago, errore mezua.
 - ▶ Kanbiorik ez badago, diru zehatza eskatu.
 - ▶ Funtzionalitatea diseinatu, makinaren arabera, egoera gehiago egon daitezkeela jakinda.
- 

Ebazpena




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().stockaEguneratu();  
        Makina.getMakina().kanbioakEguneratu();  
        if(Makina.getMakina().stockGabe())  
            Makina.getMakina().egoeraAldatu(new StockGabe());  
        else if(Makina.getMakina().kanbioGabe())  
            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());
    }
}
```



```
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(){
```

```
        //Egoeraren arabera desberdin erantzun du  
        egoera.eskatu();}
```

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

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

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

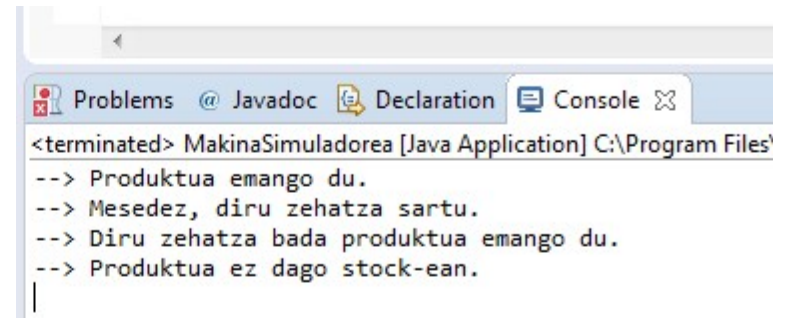
```
    public boolean stockGabe(){return stock == 0;}
```

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


Ez dago IF edo
CASErik \Rightarrow
HEDAGARRIA

Ebazpena

```
public class MakinaSimuladorea {  
  
    public static void main(String[] args) {  
        Makina.getMakina().produktuaEskatu();  
        Makina.getMakina().produktuaEskatu();  
        Makina.getMakina().produktuaEskatu();  
  
    }  
  
}
```



Ariketa: Paint

- ▶ Paint moduko aplikazio bat diseinatu
 - ▶ Tresnen barran, *marra*, *laukia* eta *borragoma* aukeratu daitezke.
 - ▶ Sagua sakatuta mantentzen badugu marrazteko eremuan, gauza desberdinak egingo dira tresnaren arabera; hots, *marra marraztu*, *laukia marraztu* edo *dagoena ezabatu*.
 - ▶ Funtzionalitate honen klase diagrama eta inplementazioa egin.
- 

STATE vs. STRATEGY

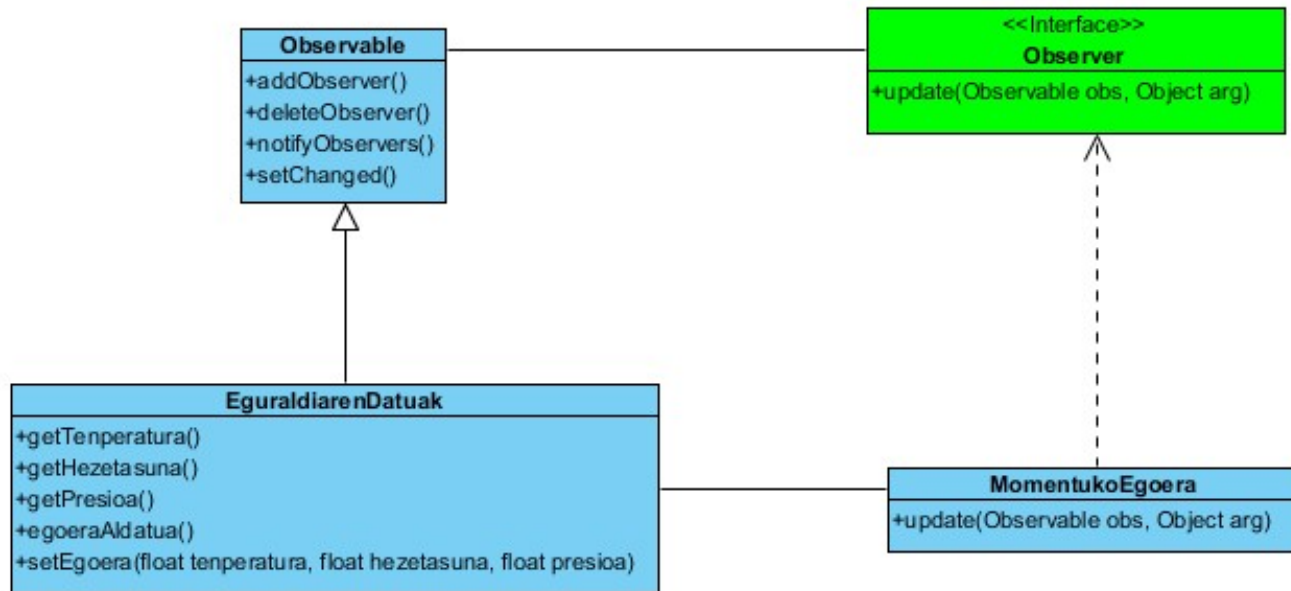
- ▶ Eskema bera dute, zein da beraien arteko desberdintasuna?
 - Bada, **ASMOA!**
- ▶ State: portaera multzoak egoera objektuetan kapsulatu.
 - Bezeroak ez daki ia ezer egoeren inguruan
- ▶ Strategy: “runtime”ean strategy objektuak aldatzea ahalbidetu.
 - Bezeroak aukeratu zer egin behar den.

Observer



Eskema orokorra

Observer: objektuen arteko “one-to-many” dependentziak definitu; objektu batek bere egoera aldatzen duenean, bere menpeko guztiei jakinaraziko die.

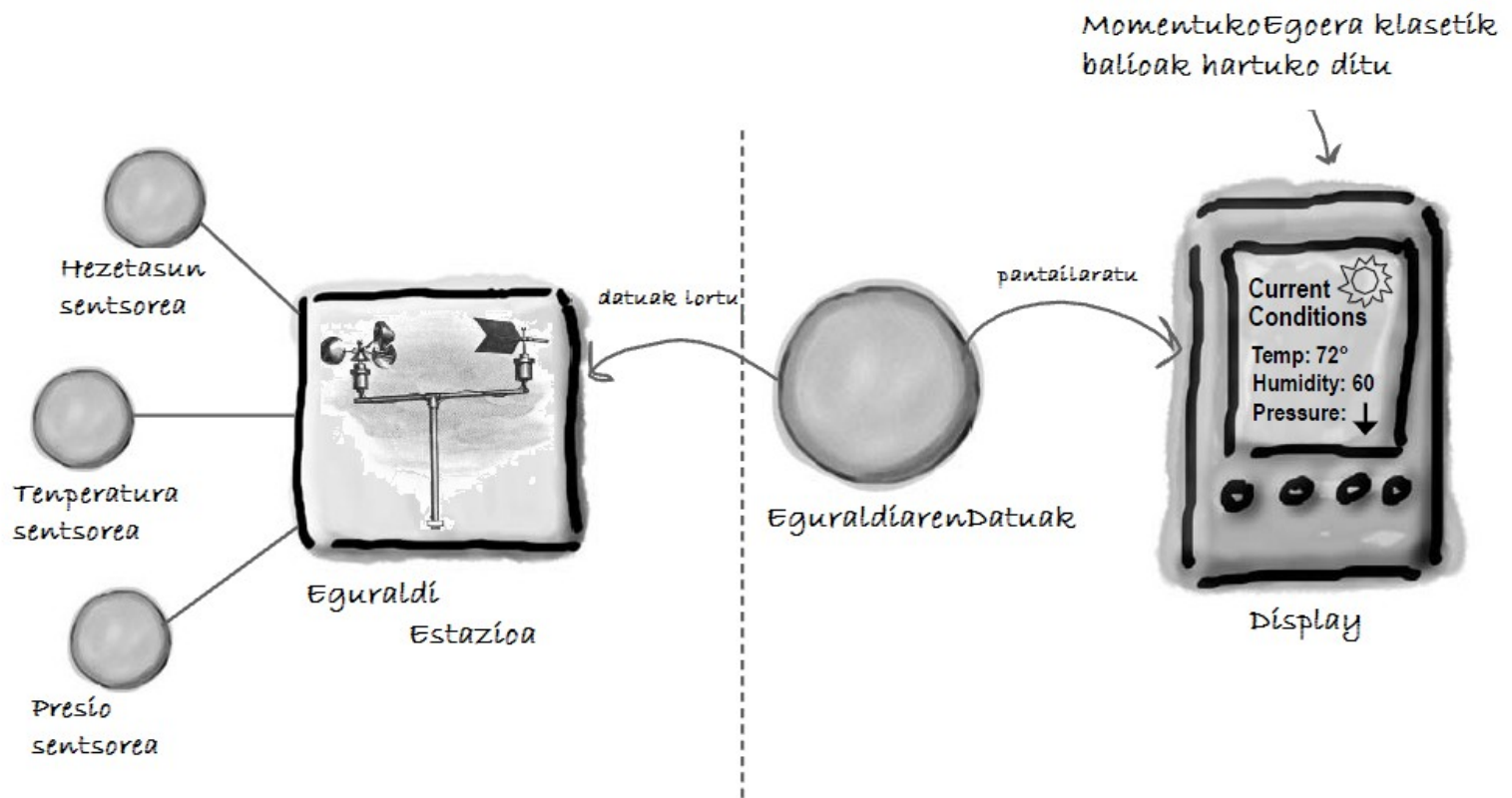


Ondorioak

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

Arazoa

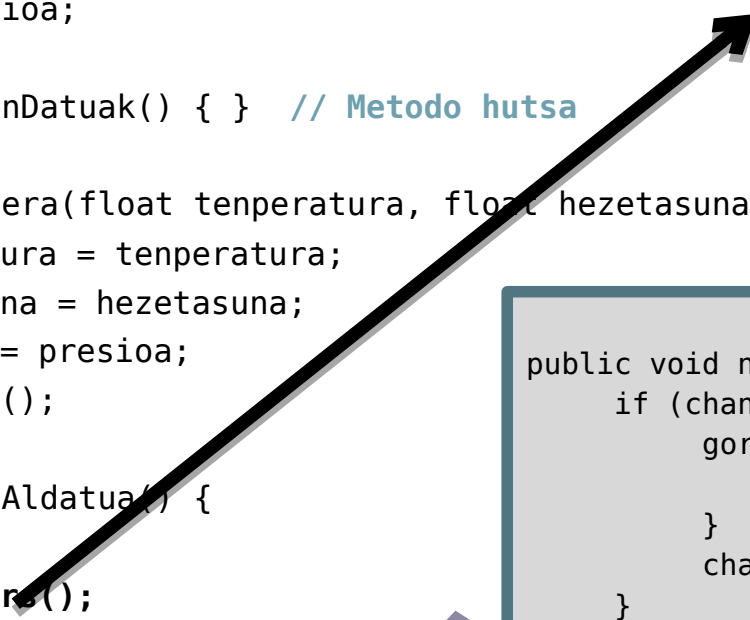
- Sistemaren klase diagrama egin, gutxienez bi display egon daitezkeela jakinda.




Ebazpena: Java

```
import java.util.Observable; //Javak implementatuta dituen liburutegiak.  
import java.util.Observer;
```

```
public class EguraldiarenDatuak extends Observable {  
    // Orain ez da beharrezkoa observer array bat gordetzea.  
    private float tenperatura;  
    private float hezetasuna;  
    private float presioa;  
  
    public EguraldiarenDatuak() { } // Metodo hutsa  
  
    public void setEgoera(float tenperatura, float hezetasuna, float presioa) {  
        this.tenperatura = tenperatura;  
        this.hezetasuna = hezetasuna;  
        this.presioa = presioa;  
        egoeraAldatua();  
    }  
    public void egoeraAldatua() {  
        setChanged();  
        notifyObservers();  
    }  
}
```



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



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

Ebazpena

```
public class MomentukoEgoera implements Observer {
    private float temperatura;
    private float hezetasuna;
    private Observable eguraldiarenDatuak;

    public MomentukoEgoera (Observable eguraldiarenDatuak) {
        this.eguraldiarenDatuak = eguraldiarenDatuak;
        eguraldiarenDatuak.registerObserver(this); // MomentukoEgoera klasea, observer
                                                    moduan gordetzen da.
    }

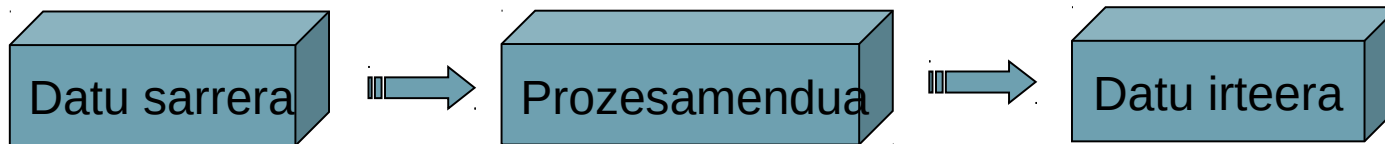
    public void update(float temperatura,float hezetasuna,float presioa){
        this.temperatura = temperatura;
        this.hezetasuna = hezetasuna;
        display();
    }

    public void display() {
        System.out.println("Momentuko egoera: " + temperatura + "gradu eta " +
            hezetasuna + "% hezetasuna");
    }
}
```

Model-View- Controller (MVC)

Model-View-Controller

Aplikazio guztietan hiru fase daude:

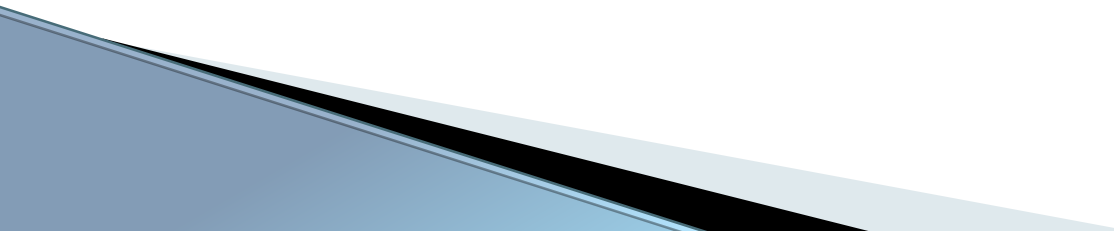


Azken horiei dagokion programazio modularra:

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

Model-View-Controller

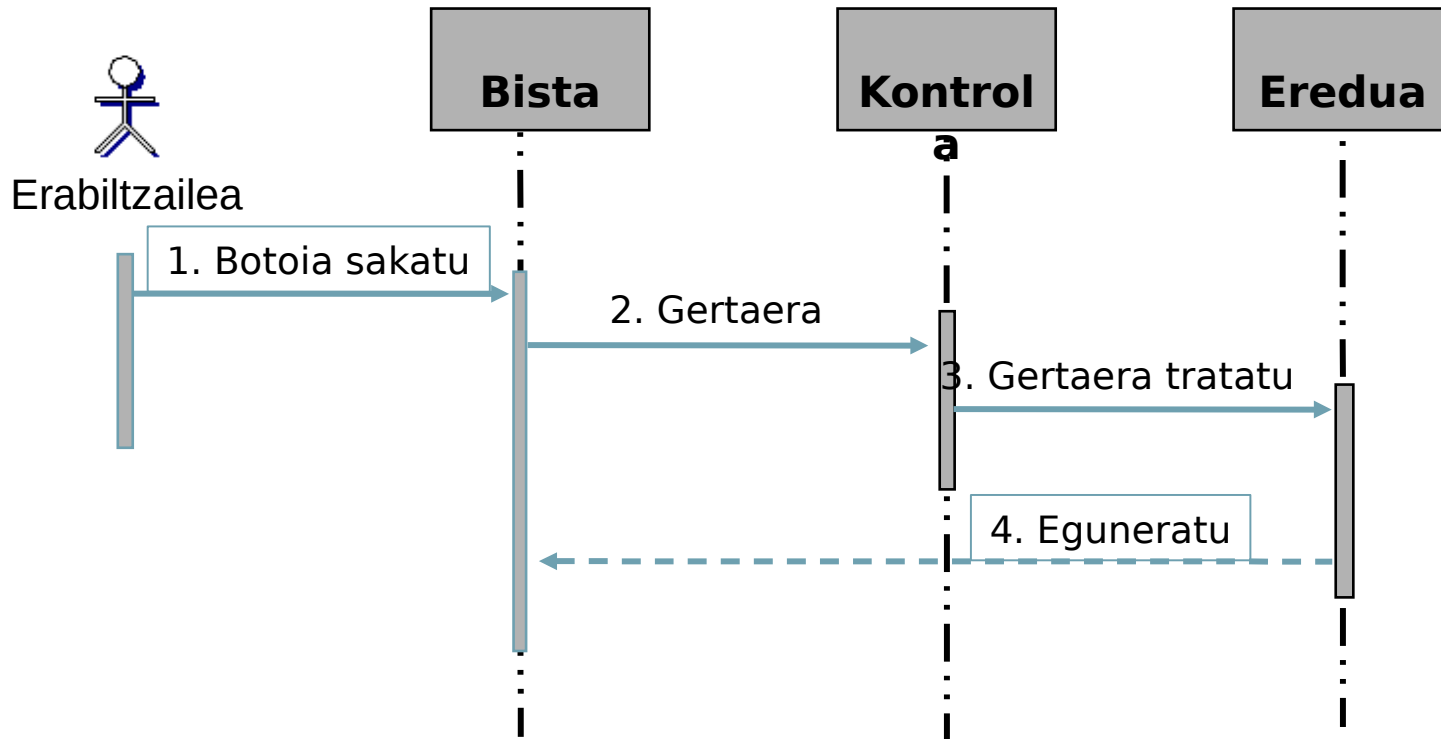
Aplikazio batetan, datuak, bista eta kontrol logika banatzen ditu:

- ▶ **Eredua:** ebatzi beharreko arazoa. Aplikazioaren informazio domeinuaren adierazpen konkretua. Domeinu logikak datuei esanahia eranstean die.
 - ▶ **Bista:** eredua/erabiltzaile elkarrekintza ahalbidetzeko interfazea
 - ▶ **Kontroladorea:** erabakiak hartzen dituen kodea. Erabiltzailearen ekintzei (gertaerei) erantzun eta ereduan aldaketak eragiten ditu.
- 

Model-View-Controller

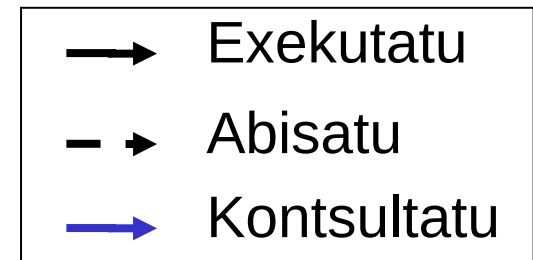
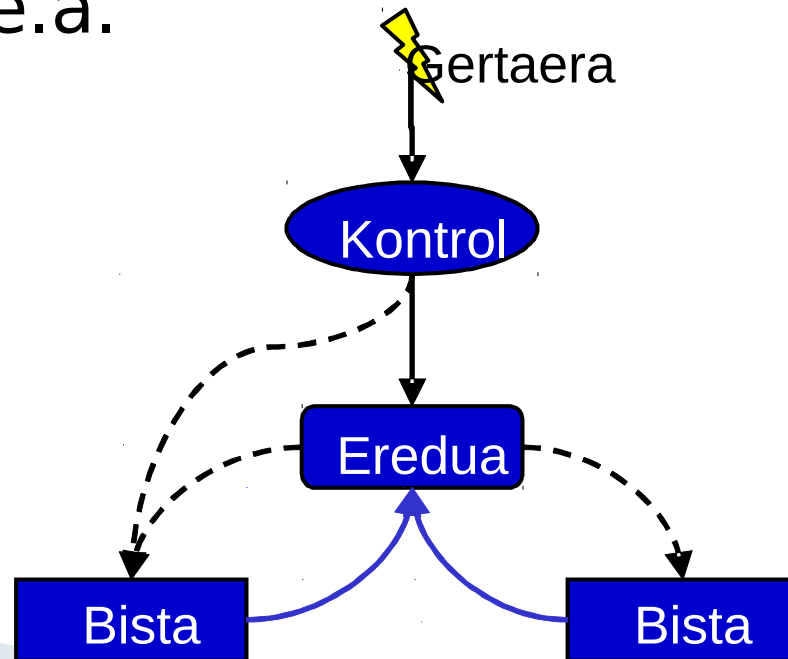
Hiru osagai horiek banatuz, hiruretako edozein aldatzeko gai izango gara, baina, besteen funtzionamendua ahalik eta gutxien ikututa. Gainera, berrerabilpena erraztuko dugu.

MVC aplikazio baten funtzionamendua



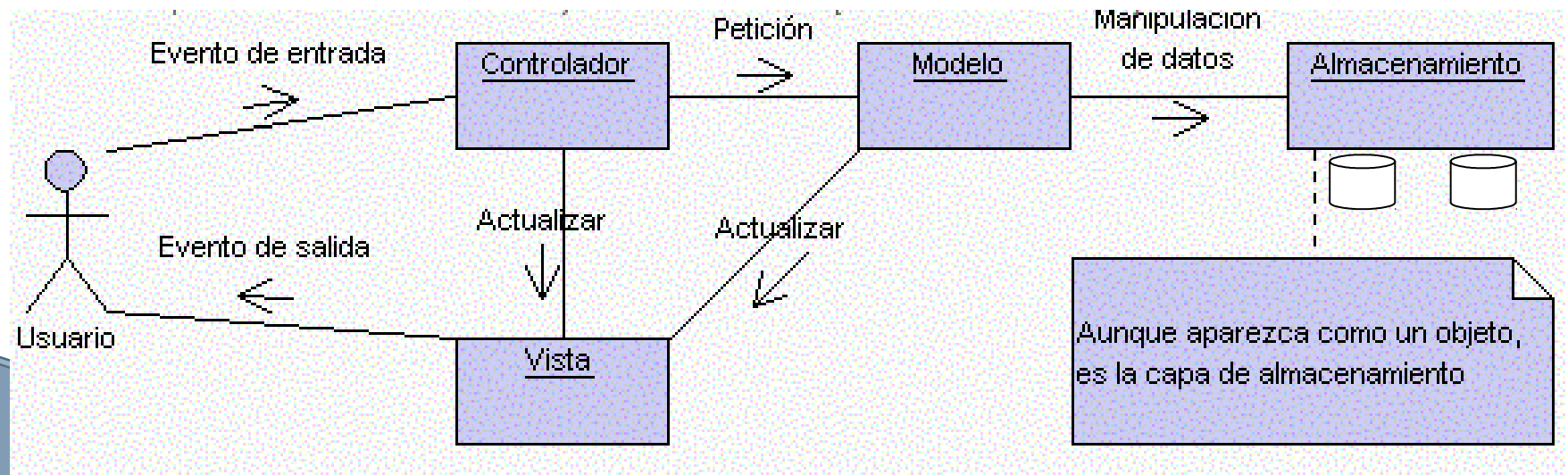
Model-View-Controller arteko menpekotasunak

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

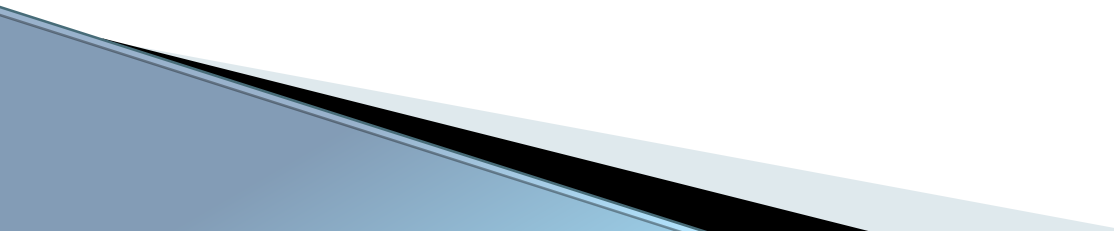


MVC aplikazio baten funtzionamendua

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
- 

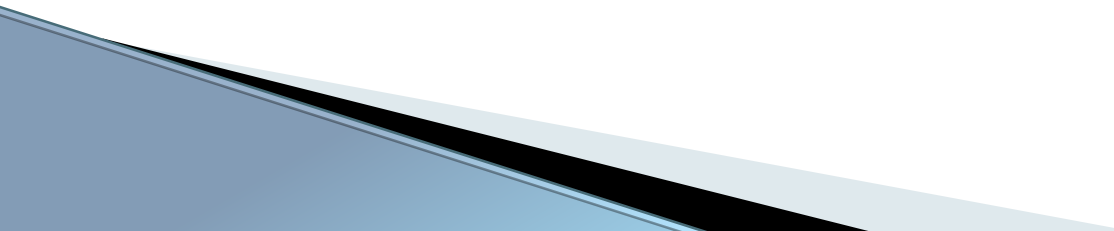
Adibidea: Dragamina

Eredua:

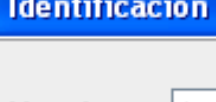
- ▶ Jokoa osatzen duten klaseak (tableroa, gelaxkak...)
- ▶ Jokuaeren erregelak tableroa marrazteko baliabide grafikoen (3Dn edo planoan) independenteak dira. Kode horrek ereduan parte hartzen du.
- ▶ Objektuei bideratutako metodologiek klase mota honetarako sarrera ematen digute, eta negozio klaseak deitzen da.

Adibidea: Dragamina

Bista:

- ▶ Jokuako aurkezpen bisuala. Interfaze grafiko bat izan ohi da, baina, testua beste programa batetik edo inpresora batekiko komunikazioa, etab. izan daiteke.
 - ▶ Jokuaren leihoez osatuta.
 - ▶ Bista ezberdinak definitu eredua aldatu barik.
- 

Adibidea: Dragamina




Identificación del usuario

Nombre:

Nivel:

OK



Lista de puntuaciones

260	<----	Bego
240	<----	Joni
150	<----	Joxean
120	<----	Mikel

OK

Adibidea: Dragamina

Kontrolatzailea:

Erabakiak hartzeko kodea da. Kode horrek ez dauka leiho bisual ezta ereduarekin arauekin zerikusirik.

Kontrolatzailearen parte dira: jokalariak bistarekin interakzionatzerakoan agertutakoan gertaerak agertzen dira, eta horiei erantzuna ematen zaie.

- ▶ Botoiren bat sakatzuz
- ▶ *Izena* eta *Maila* eremuak betez
- ▶ Tableroaren kutxatilararen baten sakatzuz.

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>