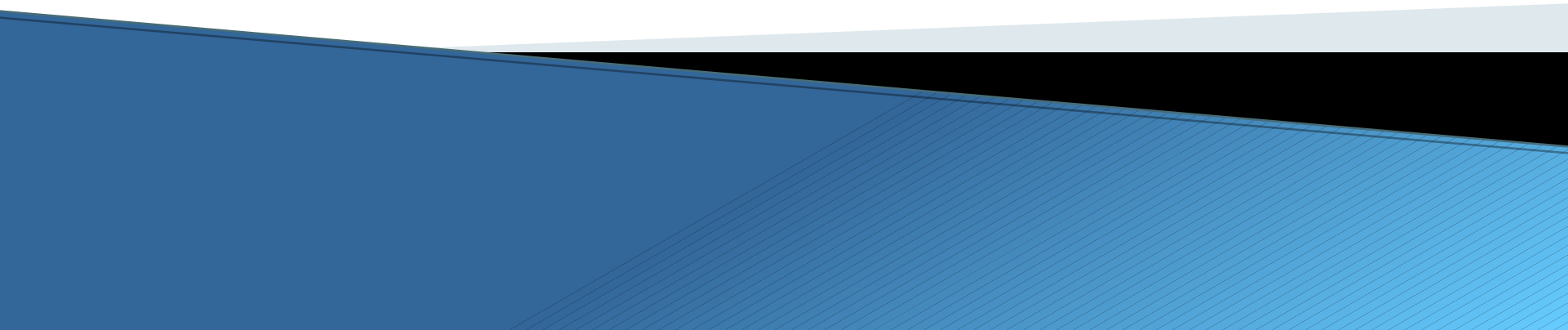
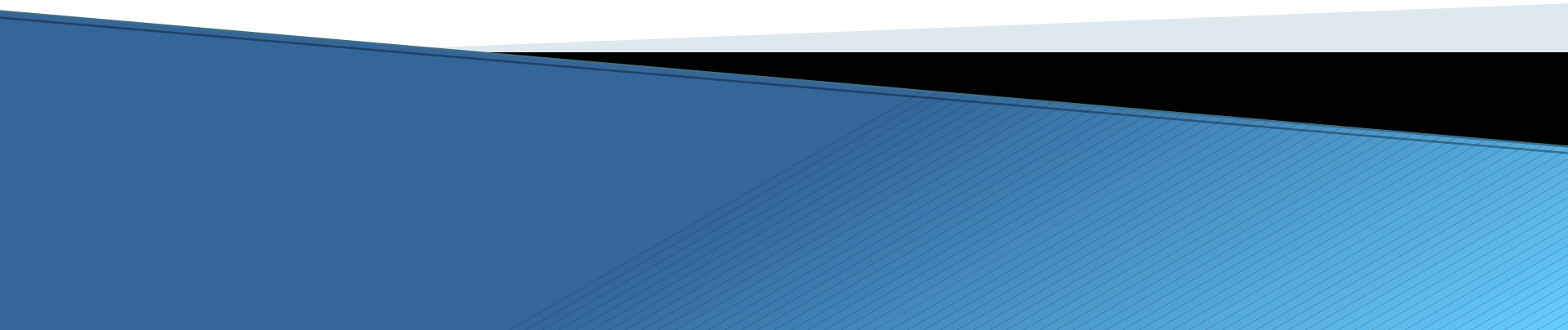


# **Diseinu Patroiak**

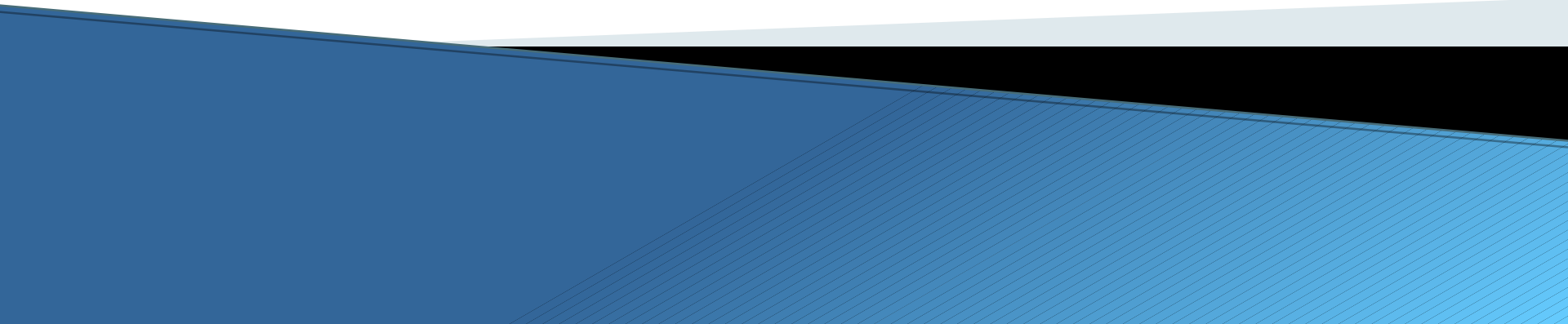
SOFTWARE INGENIARITZA



# Sortzaileak

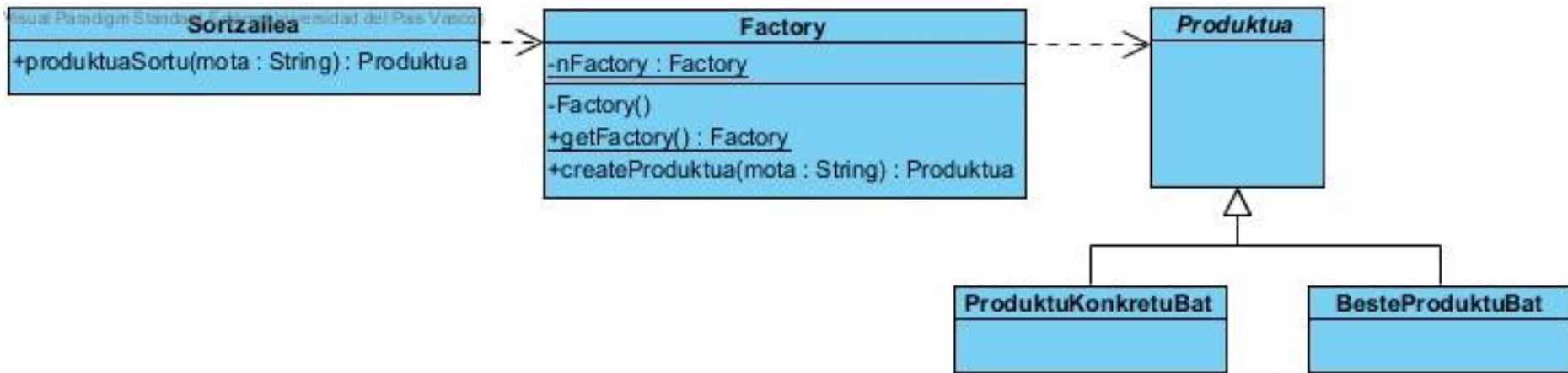


# Simple Factory



# Eskema Orokorra


**Factory:** objektuak sortzeko interfazea definitu, baina, azpiklaseen esku klaseen instantiazioaren kudeaketa



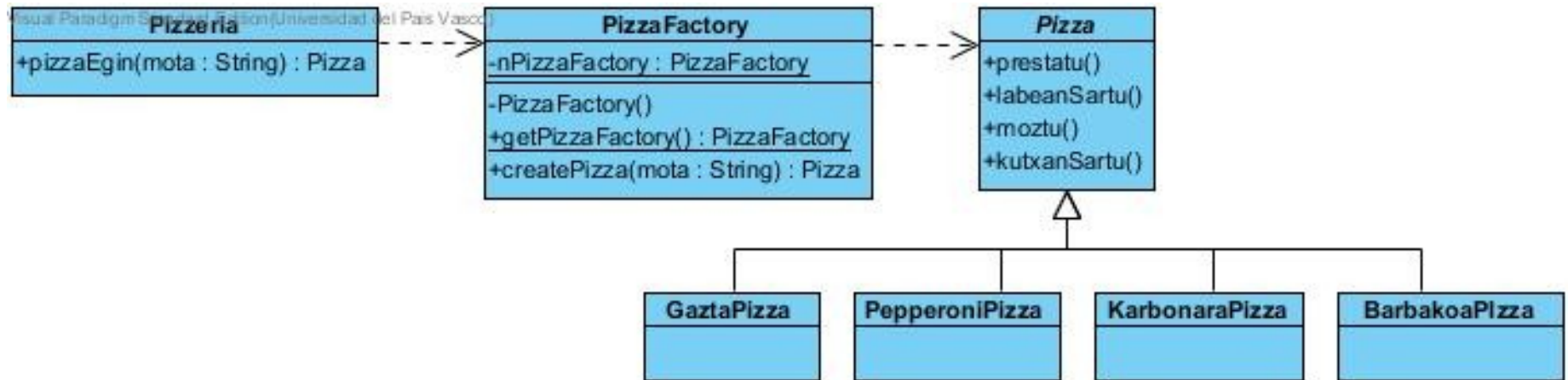
# Ezaugarriak

- ▶ Objektuen sorrera faktorian kapsulatuta
- ▶ Objektuen sorrera (faktoria) eta objektuekin lan egitea (pizzeria) banatuta
- ▶ Pizza mota berri bat sortzeko
  - Klase abstraktua hedatzeko klasea sortu
  - Faktorian bi lerro gehitu
- ▶ Objektuen sorrera kontrolatu
- ▶ Mantenketa eta hedatzea erraztu

# Arazoa

- ▶ Pizzeria batetako aplikazioan, hurrengoak saldu: *gazta, pepperoni, karbonara, barbakoa*
  - ▶ Pizza bakoitzerako: *prestatu, labean sartu, moztu eta kutxan sartu.*
  - ▶ Pizzak egiteko aplikazioaren diseinua egin, etorkizunean pizza mota gehiago egitea posible dela kontutan hartuz.
- 

# Ebazpena



# Ebazpena

```
public class Pizzeria {
    public Pizzeria(){
    }
    public Pizza pizzaEgin (String mota){
        Pizza nirePizza = PizzaFactory.getPizzaFactory().createPizza(mota);
        nirePizza.prestatu();
        nirePizza.labeanSartu();
        nirePizza.moztu();
        nirePizza.kutxanSartu();
        return nirePizza;
    }
    public static void main(String [ ] args){
        Pizzeria nirePizzeria = new Pizzeria();

        Pizza bbPizza = nirePizzeria.pizzaEgin("Barbakoa");
        System.out.println("Pizza eginda dago eta " + bbPizza.getClass().toString() + "
                                motakoa da!");
    }
}
```



# Ebazpena

```
public class PizzaFactory {  
  
    private static PizzaFactory nPizzaFactory;  
  
    private PizzaFactory (){}  
  
    public static PizzaFactory getPizzaFactory(){  
        if (nPizzaFactory == null) {nPizzaFactory = new PizzaFactory();}  
        return nPizzaFactory;  
    }  
  
    public Pizza createPizza (String mota){  
        Pizza nirePizza = null;  
        if(mota == "Gazta"){nirePizza = new GaztaPizza();}  
        else if(mota == "Pepperoni"){nirePizza = new PepperoniPizza();}  
        else if (mota == "Karbonara"){nirePizza = new KarbonaraPizza();}  
        else if (mota == "Barbakoa"){nirePizza = new BarbakoaPizza();}  
        return nirePizza;  
    }  
}
```

# Ebazpena

```
public abstract class Pizza {  
  
    public Pizza(){}  
    public void prestatu(){System.out.println("Pizza prestatu da.");}  
    public void labeanSartu(){System.out.println("Pizza labean sartu da.");}  
    public void moztu(){System.out.println("Pizza moztu da.");}  
    public void kutxanSartu(){System.out.println("Pizza kutxan sartu da.");}  
}
```

```
public class BarbakoaPizza extends Pizza{  
    public BarbakoaPizza(){}  
}
```

....



# Ariketa: Arkanoid

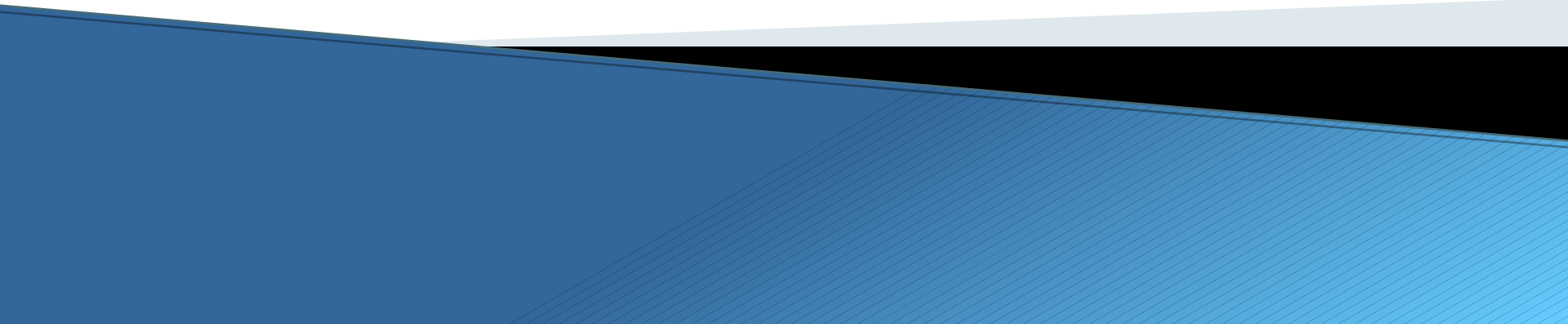


- ▶ Arkanoid jokuan adreilu horma bat suntsitu behar da (suntsiezinak ez diren bitartean), pilota bat adreiluetan errebote eraginez.
- ▶ Adreiluak mota ezberdinekoak izan daitezke: 1, 2 edo hiru kolperen ondoren puskatzen direnak.

# Ariketa: Arkanoid

- ▶ Eskatzen da:
  - Jokoaren diseinua (Klase Diagrama)
  - Jokoaren horma sortzen duen zatiaren inplementazioa (hormaren adreiluen mota ausaz erabakitzen da).

# Egiturazkoak



# Facade



# Zer da akoplamendua?

Klaseek beren artean duten **dependentzia maila** da. Zenbat eta akoplamendu txikiagoa, orduan eta eragin gutxiago izango dute sistemako aldaketek gure programan.

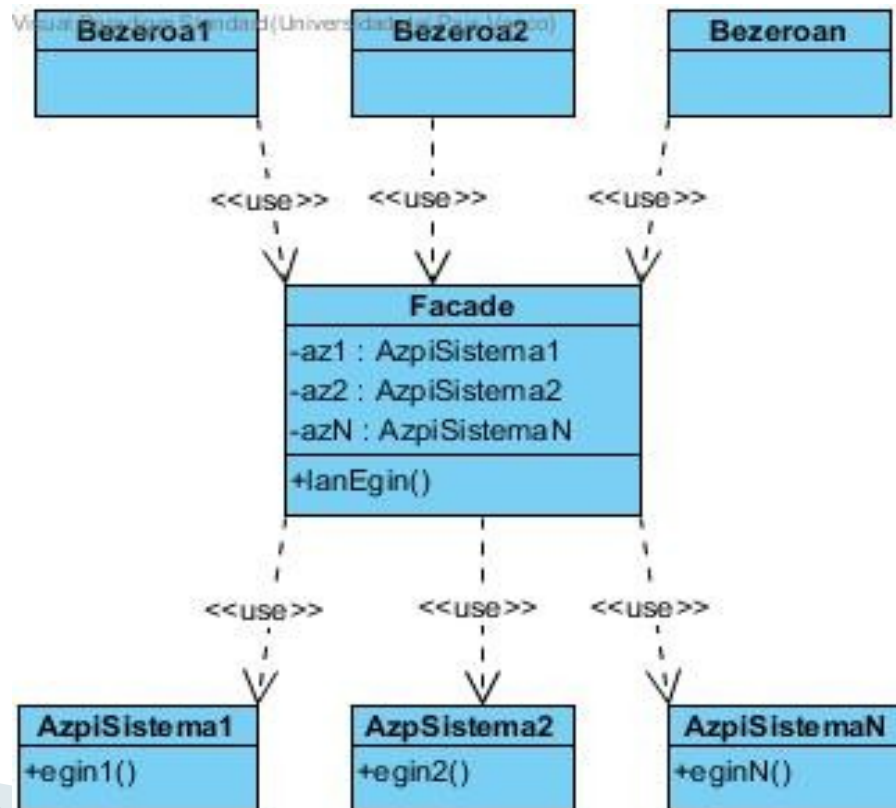
# Ezagutza minimoaren printzipioa

- ▶ Akoplamendu ahula (Loose coupling):
  - Klase batek berarekin elkarrekintza estuan daudenak soilik ezagutu
  - Klase batek bere “lagunekin” soilik berba, ez “arrotzekin”
- ▶ Helburua: akoplamendua ahalik eta gehien murriztu




# Eskema Orokorra

**Facade:** azpisistema multzo baten inertfazei interfaze bateratua ezarri; hau da, bezeroari maila altuko interfazea eskaini, azpisistemak erabilterraza goak bihurtzeko



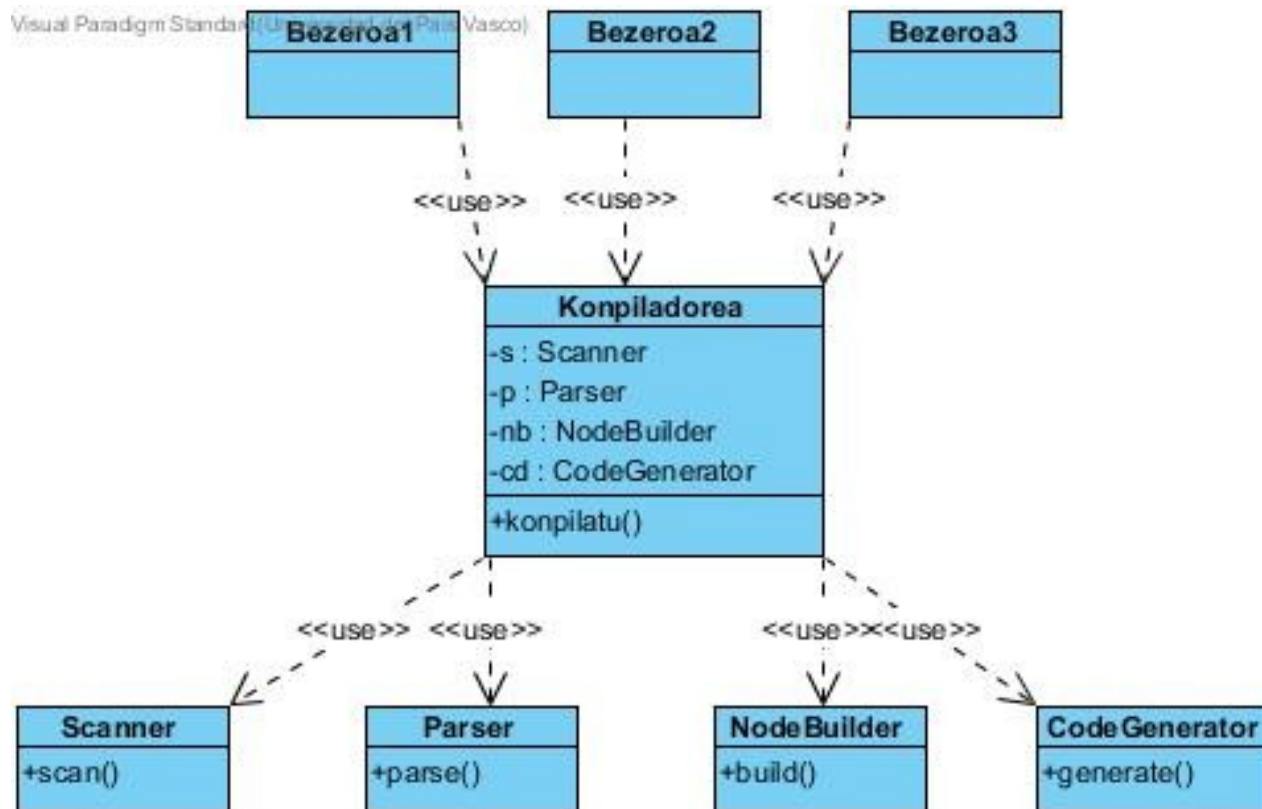
# Ezaugarriak

- ▶ Bezeroa sistematik isolatu
  - ▶ Bezero/azpisistemen akoplamendu ahula
  - ▶ Azpisistemak bezeroarentzat eskuragarri, behar izanez gero
  - ▶ Sistema geruza banatu
  - ▶ Kontuan izan: bezeroek azpisistema desberdinak erabiliz gero, facade desberdinak
- 

# Arazoa

- ▶ Java konpiladore baten diseinua
- ▶ Konpilatzeko, azpisistema desberdinak:
  - Scanner-ak programa irakurri
  - Parser-ak prozesatu
  - NodeBuilder-ak zuhaitza sortu
  - CodeGenerator-ak bytecode-a sortu
- ▶ Konpiladorearen klase diagrama egin, etorkizunean azpisistemak aldatu daitezkeela kontutan hartuz.

# Ebazpena



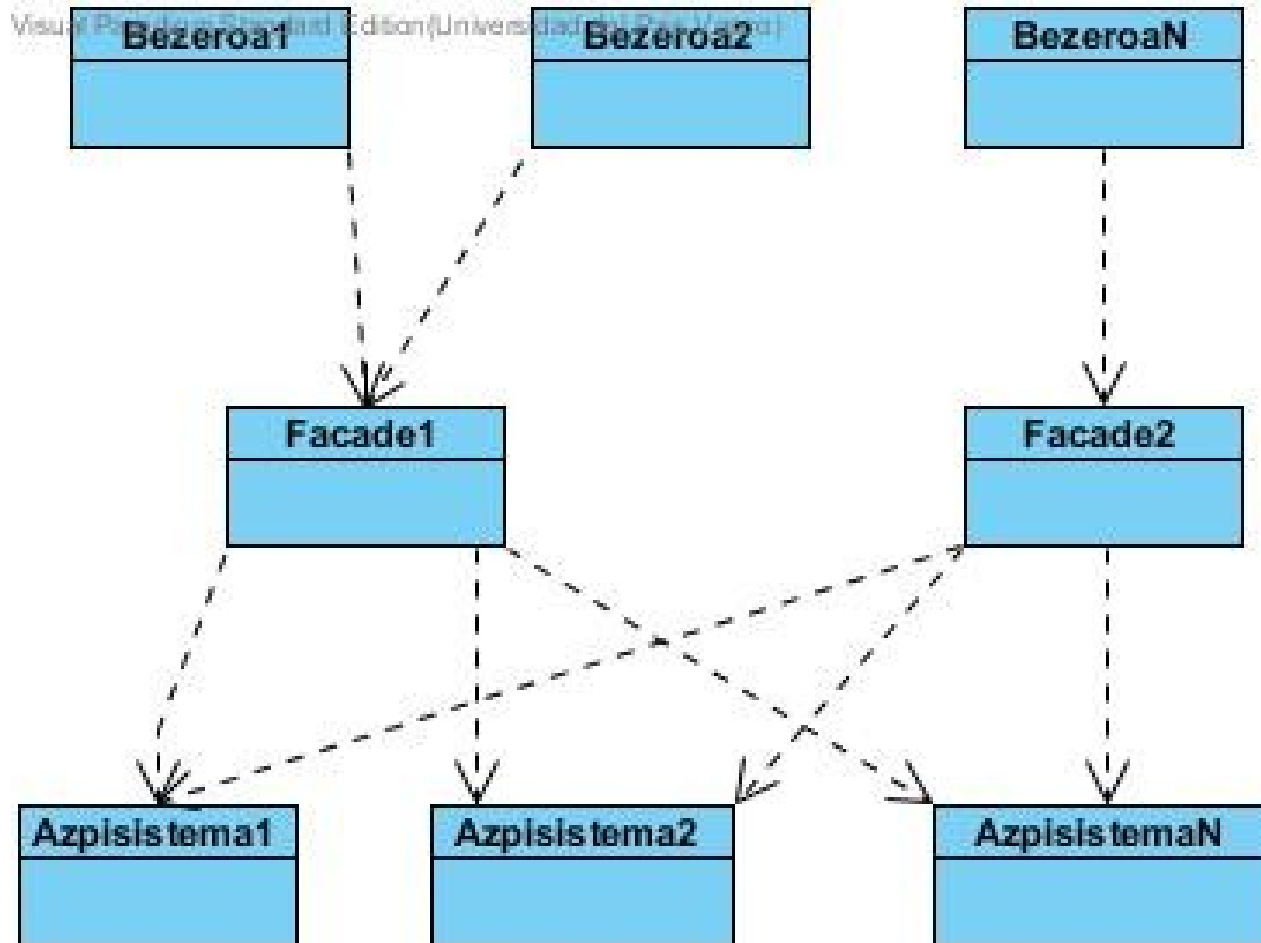
\* Sinplifikatze aldera, ez dira parametroak diagraman gehitu

# Ebazpena

```
public class Konpiladorea {  
    private static Konpiladorea nKonpiladorea;  
    private Scanner scanner;  
    private Parser parser;  
    private NodeBuilder nodeBuilder;  
    private CodeGenerator codeGenerator;  
  
    private Konpiladorea () {  
        scanner = new Scanner();  
        parser = new Parser();  
        nodeBuilder = new NodeBuilder();  
        codeGenerator = new CodeGenerator();  
    }  
  
    public static Konpiladorea getKonpiladorea(){...}  
  
    public void compile(){  
        scanner.scan();  
        parser.parse();  
        nodeBuilder.build();  
        codeGenerator.generate();  
    }  
}
```

Azpisistemak  
isolatzen ditu

# Orokortuz



# Ariketa: Multimedia Gela

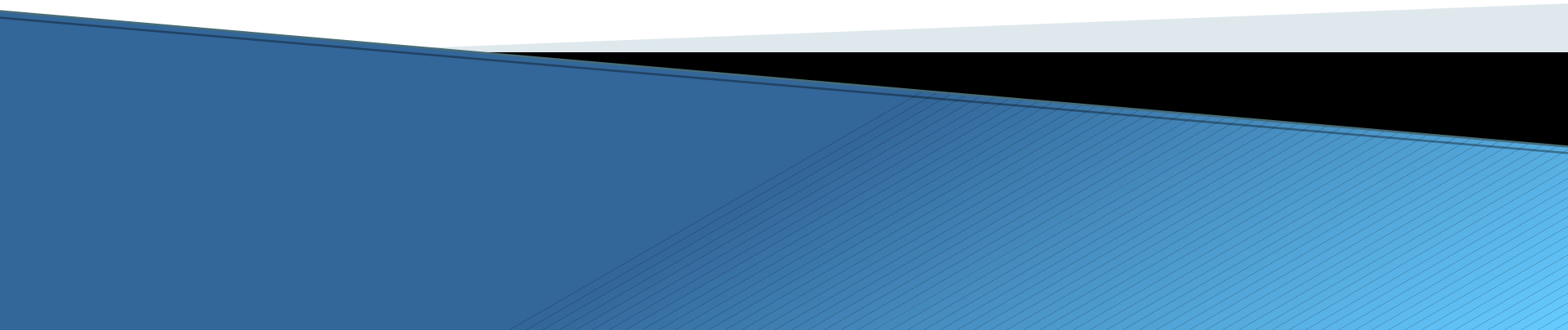
- ▶ Multimedia gela bat kudeatzeko sistema inplementatu.
- ▶ Gelan bi motatako ekitaldiak: *pelikula emanaldiak* eta *hitzaldiak*.
- ▶ Bi kasuetan, *pantaila jaitsi* eta *proiektorea piztu*
- ▶ Pelikula emanaldietan, gainera: *proiektorea DVD moduan jarri, DVD-a piztu, bozgorailuak piztu, bere bolumena finkatu, diskoa sartu eta diskoa martxan jarri.*

# Ariketa: Multimedia Gela

- ▶ Hitzaldietan, gainera: *proiektorea PC moduan jarri, ordenagailua piztu eta aurkezpena martxan jarri*
- ▶ Sistemaren klase diagrama eta ekitaldi mota bakoitza kudeatzeko zatiaren inplementazioa

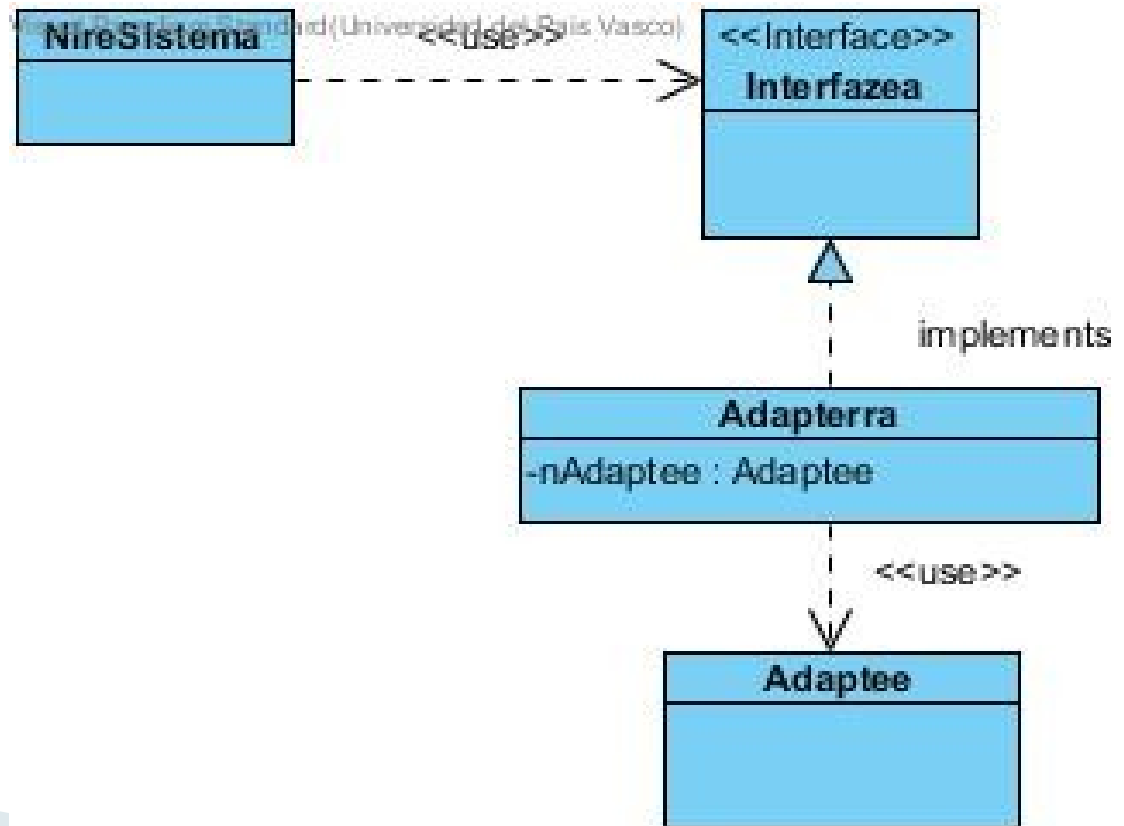


# Adapter




# Eskema Orokorra

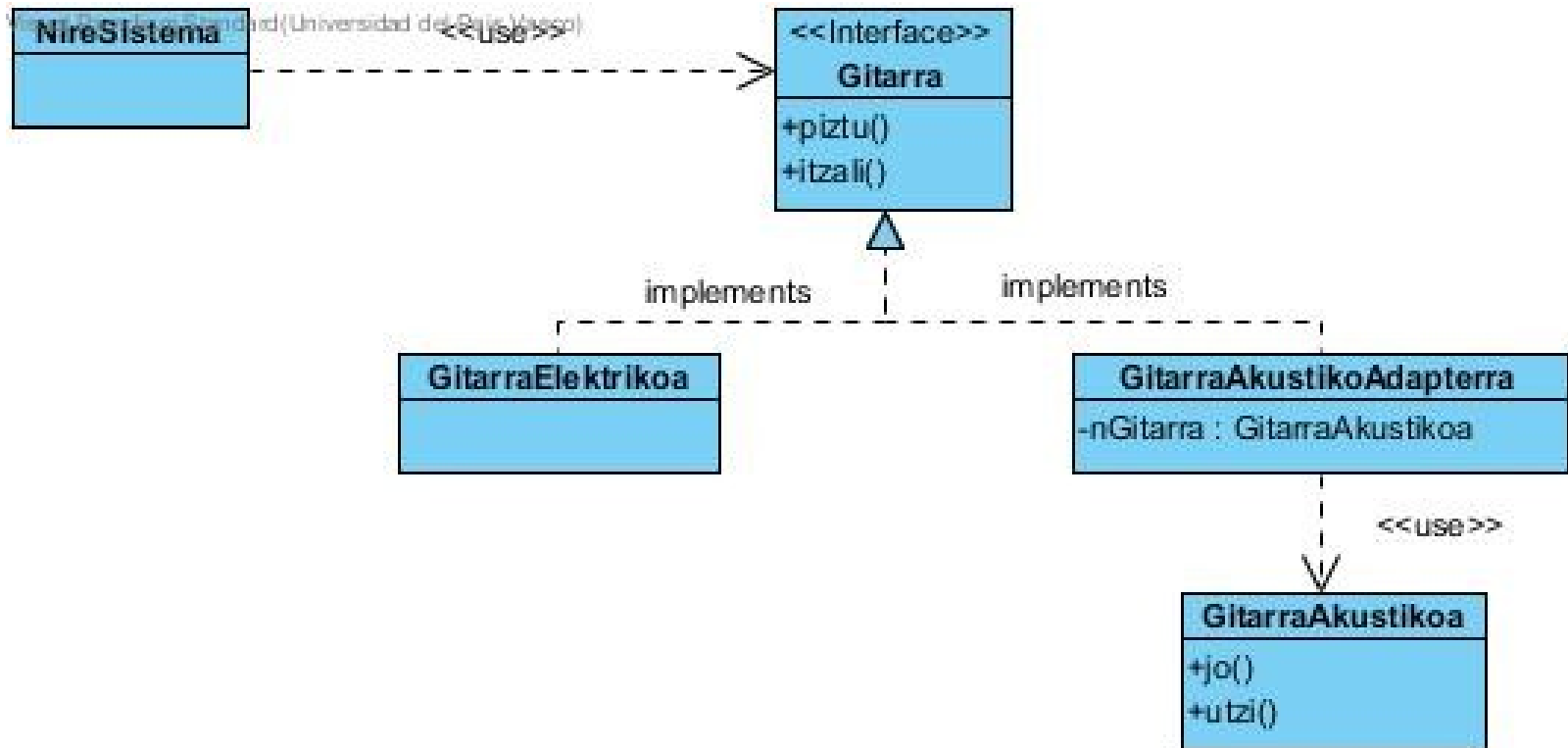
**Adapter:** klase baten interfazea bezeroak esperotako interfazera bihurtzen du; hau da, interfaze bateraezinei elkarrekin lan egiteko aukera ematen die



# Arazoa

- ▶ Musika tresnak simulatzeko sistema dugu.
  - ▶ Gitarrak simulatzeko interfaze bat dago, *piztu* eta *itzali* metodoekin.
  - ▶ Interfaze hori inplementatzeko, gitarra elektriko bat dugu.
  - ▶ Beste sistema bateko gitarra akustikoa berrerabili nahi dugu, baina, *jo* eta *utzi* metodoak ditu.
  - ▶ Sistemaren klase diagrama egin, berrerabilgarria izan behar duela kontutan hartuz.
- 

# Ebazpena



# Ebazpena

```
public interface Gitarra {  
    public void piztu();  
    public void itzali();  
}
```

Klase abstraktu bat  
ere izan daiteke

```
public class GitarraElektrikoa implements Gitarra {  
    public void piztu(){...}  
    public void itzali () {...}  
}
```

```
public class GitarraAkustikoa {  
    public void jo(){...}  
    public void utzi(){...}  
}
```

Ez du  
GitarraAkustikoa  
aldatzen

```
public class GitarraAkustikoAdapterra implements Gitarra {  
    private GitarraAkustikoa gitarraAkustikoa = new GitarraAkustikoa();  
  
    public void piztu(){gitarraAkustikoa.jo();}  
    public void itzali(){gitarraAkustikoa.utzi();}  
}
```

# Ariketa: Motorrak

- ▶ Audi kotxeak kontrolatzeko sistema
- ▶ Motorrak hiru operazio: *piztu*, *azeleratu* eta *itzali*.
- ▶ Motore elektrikoak kudeatzeko sistema beste enpresa bati erosi diogu. Kasu horretan, motoreek *konektatu*, *aktibatu*, *azkartu*, *gelditu* eta *deskonektatu* operazioak dituzte.

# Ariketa: Motorrak

- ▶ Hurrengoa eskatzen da:
  - Sistemaren diseinua (Klase Diagrama)
  - Gure sisteman motor elektrikoak sartzea ahalbidetuko duen zatiaren inplementazioa.

# Composite

The bottom of the slide features a decorative graphic consisting of several overlapping geometric shapes. On the left, there is a solid dark blue trapezoidal shape. To its right, a black horizontal band is visible. Further right, a light blue trapezoidal shape overlaps the black band. The bottom right corner is filled with a series of parallel, diagonal light blue lines.

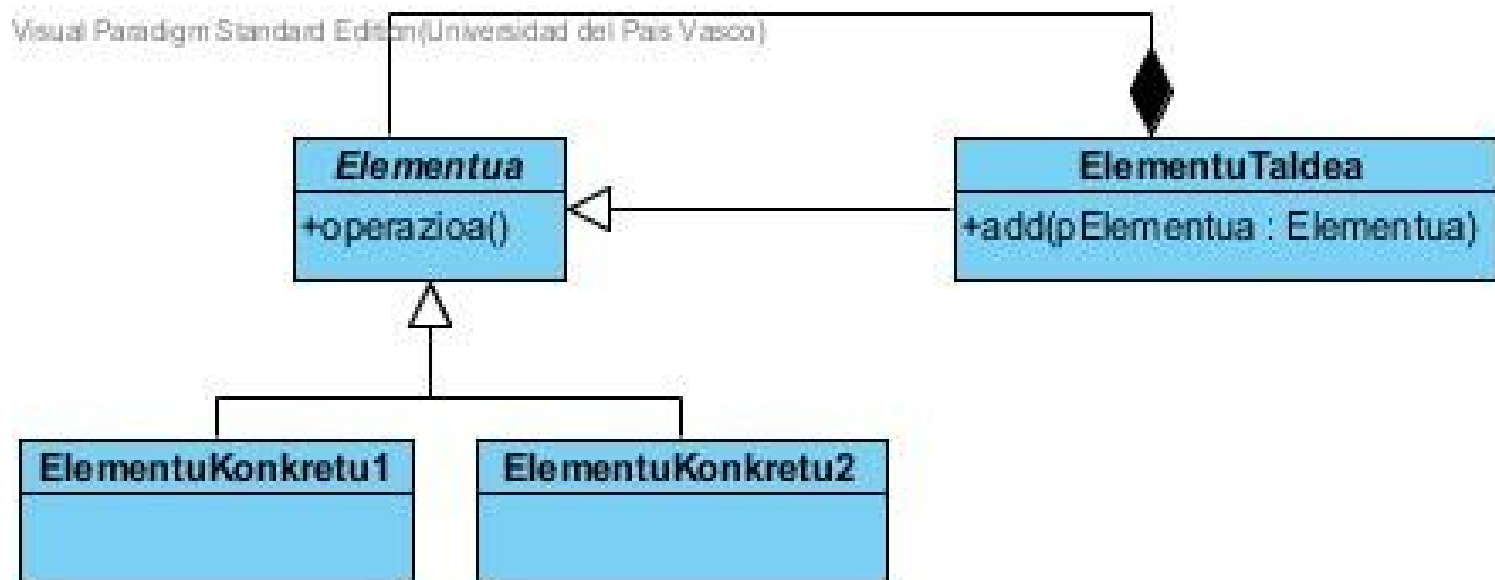


# Ondorioak

- ▶ Berrerabilgarritasuna hobetu
- ▶ Hedapena erraztu
- ▶ Adapterrak interfazeak kapsulatu
  - Bezeroa interfazetik desakoplatu
  - Interfazea aldatuz gero, bezeroak ez du ikusten
- ▶ Adaptee-a ez da ikutzen

# Eskema Orokorra

**Adapter:** “part-whole” hierarkiak errepresenatzeko aldera, objektuak zuhaitz egituretan osatzea ahalbidetzen du. Bezeroei banakako objektuak eta konposatuak uniformeki erabiltzen uzten die.



# Ondorioak

- ▶ Banakako elementuak (hostoak) eta konposatuak (nodoak) zuhaitz egitura berean txertatu
  - “Part-whole” hierarkiak
- ▶ Objektu guztiek interfaze bera
  - Nodo zein hosto, era berean tratatu

# Arazoa

- ▶ Aplikazio baten elementu grafikoen informazioa biltzeko klaseak behar ditugu. Adibidez, *biribilak*, *laukiak* eta *hirukiak*.
- ▶ Irudi multzoak tratatu behar ditugu. Programak zenbait irudi batera lantzeko aukera egin behar du, objektu bakarra balitz bezala; pantailan zehar mugitzeko, koloreztatzeko edo berdimentsionatzeko

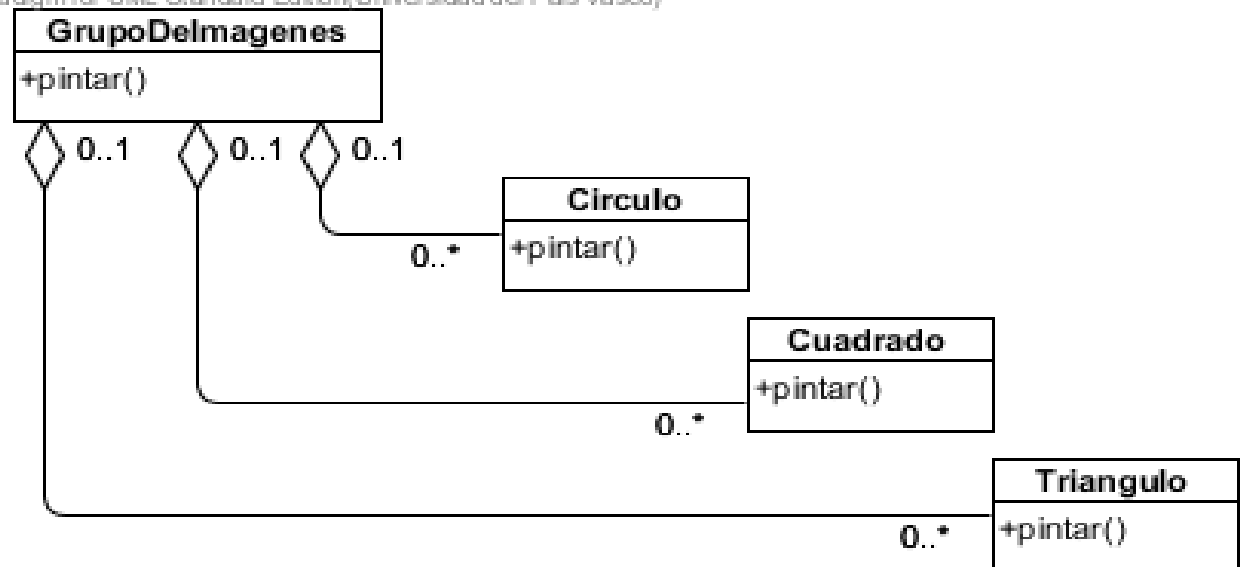
# Arazoa

- ▶ Aplikazioa diseinatzeko, figura mota bakoitzarentzat klase bat definitu daiteke, dagokion *marraztu()* metodoarekin.
- ▶ Baina, nola definitu irudi multzo bat irudi bakarra balitz bezala kudeatzeko?

# Ebazpena

- Hierarkian oinarritutako hurbilketa batek egokia dirudi, hurrengoekin osatutako:
  - Irudi multzoak: irusi soilak atributu legez
  - Irudi soila

Visual Paradigm for UML Standard Edition (Universidad del País Vasco)

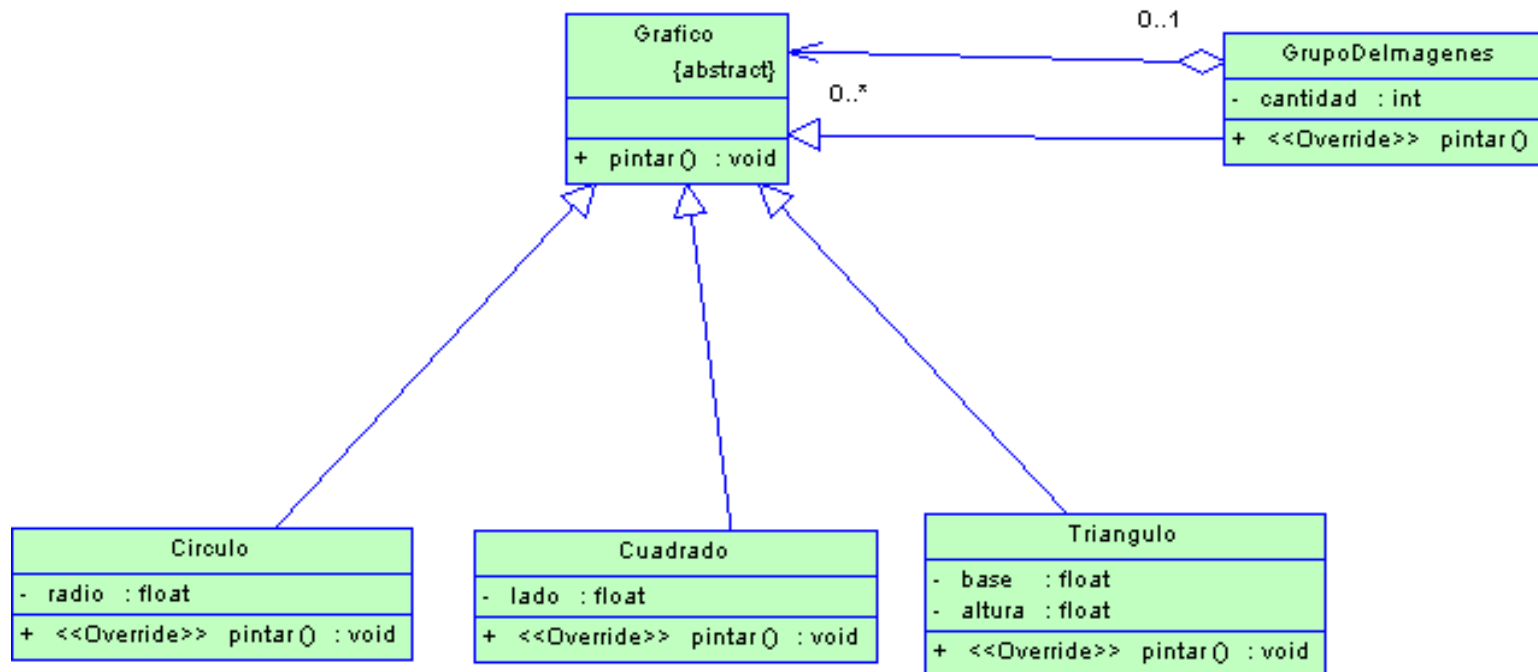


# Ebazpena

## ► Arazoak

- Soluzioa ez da eskalagarria: gehitu beharreko irudi grafiko mota berri bakoitzeko, atributu berri bat
- Irudi grafiko soilak ez du kontuan hartzen irudi talde baten parte dela.

# Ebazpena



Irudi talde batek Grafiko kolekzio bat du. Grafiko horiek Zirkuluak, Laukiak edo beste irudi talde bat izan daitezke



# Ebazpena

```
import java.util.List;
import java.util.ArrayList;

public interface Grafikoa { //grafikoa marraztu
    public void marraztu();
}
/** "Composite" */
class IrudiTaldea implements Grafikoa {
    //Irudi grafikoen kolekzioa
    private List<Grafikoa> mChildGraphics = new
    ArrayList<Grafikoa>();
    //Irudi grafikoak marrazten ditu
    public void marraztu() {
        for (Grafikoa grafikoa : mChildGraphics) {grafikoa.marraztu();}
    }
    //Irudi grafikoa konposaketari gehitzen zaio.
    public void add(Grafikoa grafikoa) {
        mChildGraphics.add(grafikoa);
    }
}
```

# Ebazpena

```
/** "Hosto klaseak" */  
class Hirukia implements Grafikoa {  
  
    //Irudi grafikoa marrazten du  
    public void marraztu() {  
        System.out.println("Hirukia");  
    }  
}
```

# Ebazpena

```
/** Bezeroa*/  
public class Proba {  
  
    public static void main(String[] args) {  
        //Lau hiruki hasieratu  
        Hirukia hirukia1 = new Hirukia();  
        Hirukia hirukia2 = new Hirukia();  
        Hirukia hirukia3 = new Hirukia();  
        Hirukia hirukia4 = new Hirukia();  
  
        //Hiru irudi grafiko konposatu hasieratu  
        IrudiTaldea talde1 = new IrudiTaldea();  
        IrudiTaldea talde2 = new IrudiTaldea ();  
        IrudiTaldea talde3 = new IrudiTaldea ();  
    }  
}
```


# Ebazpena

```
//Irudi grafikoen konposaketa
    talde1.add(hirukia1);
    talde1.add(hirukia2);
    talde1.add(hirukia3);

    talde2.add(hirukia4);

    talde3.add(talde1);
    talde3.add(talde2);

    //grafiko osoa marraztu (lau bider "Hirukia" string-a).
    talde3.marraztu();
}
}
```



# Ariketa

- ▶ Laborategian ikusitako Swing liburutegiko osagai eta edukiontzien klase diagrama egin.

# 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:  
<https://cutt.ly/vrTamMP>  
<http://hillside.net/patterns/books/>  
<http://www.javacamp.org/designPattern/>  
<http://www.dofactory.com/net/design-patterns>