Diseinu Patroiak

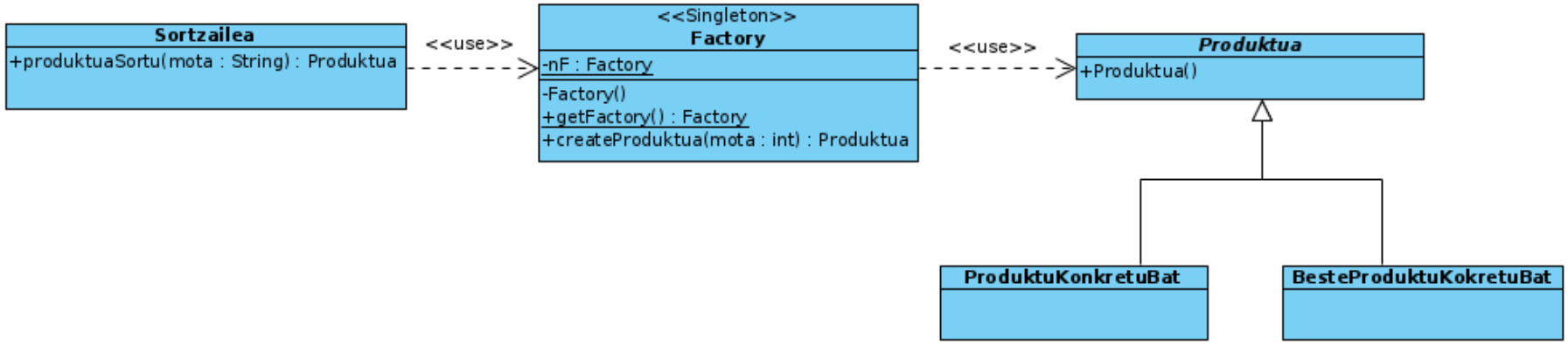
SOFTWARE INGENIARITZA

Sortzaileak

Simple Factory

Eskema Orokorra

Factory: objektuak sortzeko interfazea definitu, baina, azpiklaseen esku klaseen instanziazioaren kudeaketa.



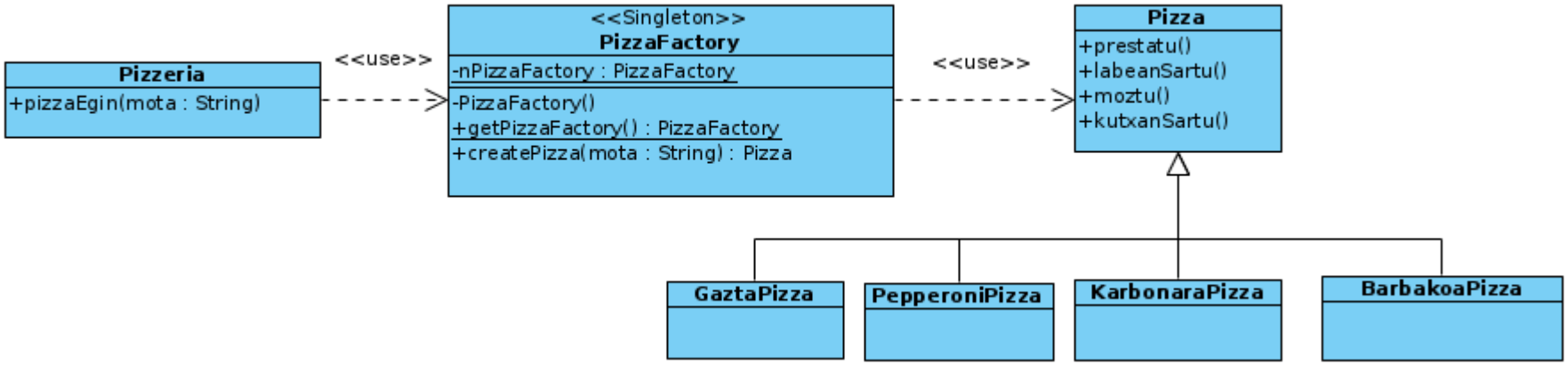
Ezaugarriak

- Objektuen sorrera faktorian kapsulatuta
- Faktoria eta objektuekin lan egitea, banatuta
- Objektu 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...*
- Pizza bakoitzerako: *prestatu, labean sartu, moztu* eta *kutxan sartu*.
- Pizzak egiteko aplikazioaren diseinua egin, etorkizunean pizza mota gehiago egitea posible dela kontutan hartuz.

Ebazpena


Pizzeria
+pizzaEgin(mota : String)
<<use>>
<<Singleton>>
PizzaFactory
-nPizzaFactory : PizzaFactory
-PizzaFactory()
+getPizzaFactory() : PizzaFactory
+createPizza(mota : String) : Pizza
<<use>>
Pizza
+prestatu()
+labeanSartu()
+moztu()
+kutxanSartu()
GaztaPizza
PepperoniPizza
KarbonaraPizza
BarbakoaPizza

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;
    }
}
```

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;
    }
}
```

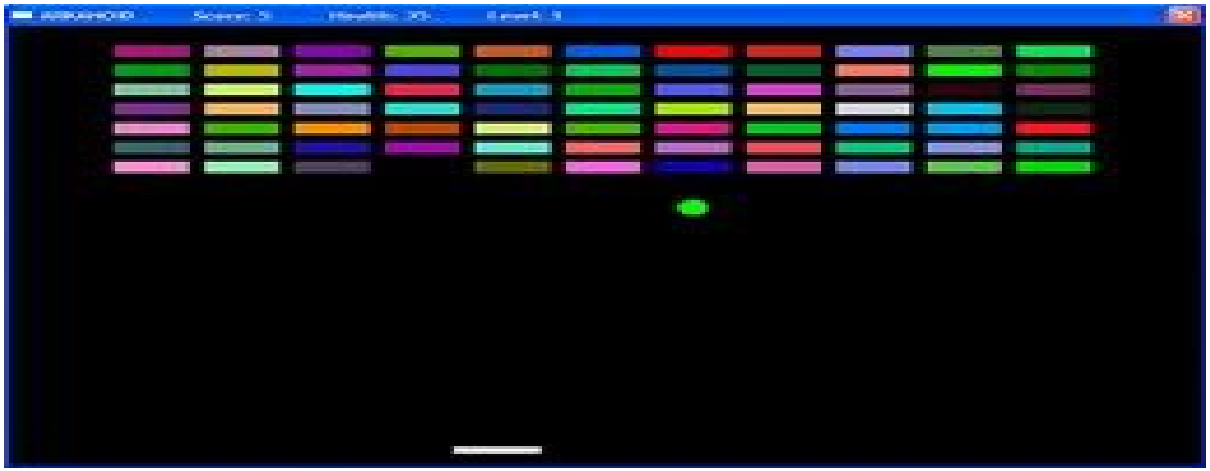
Sarrera parametroaren arabera, pizza mota guztiak sortzeko gai

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(){ ... }}
public class PeperonniPizza extends Pizza{ public PepperoniPizza(){ ... }}
public class GaztaPizza extends Pizza{     public GaztaPizza(){ ... }}
public class KarbonaraPizza extends Pizza{ public KarbonaraPizza(){ ... }}
```

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

- Hurrengoa burutu ezazu:
  - Jokoaren diseinua (Klase Diagrama)
  - Jokoaren horma sortzeko inplementazioa. Hormaren adreilu mota ausaz erabaki.

Egiturazkoak

Facade

Zer da akoplamendua?

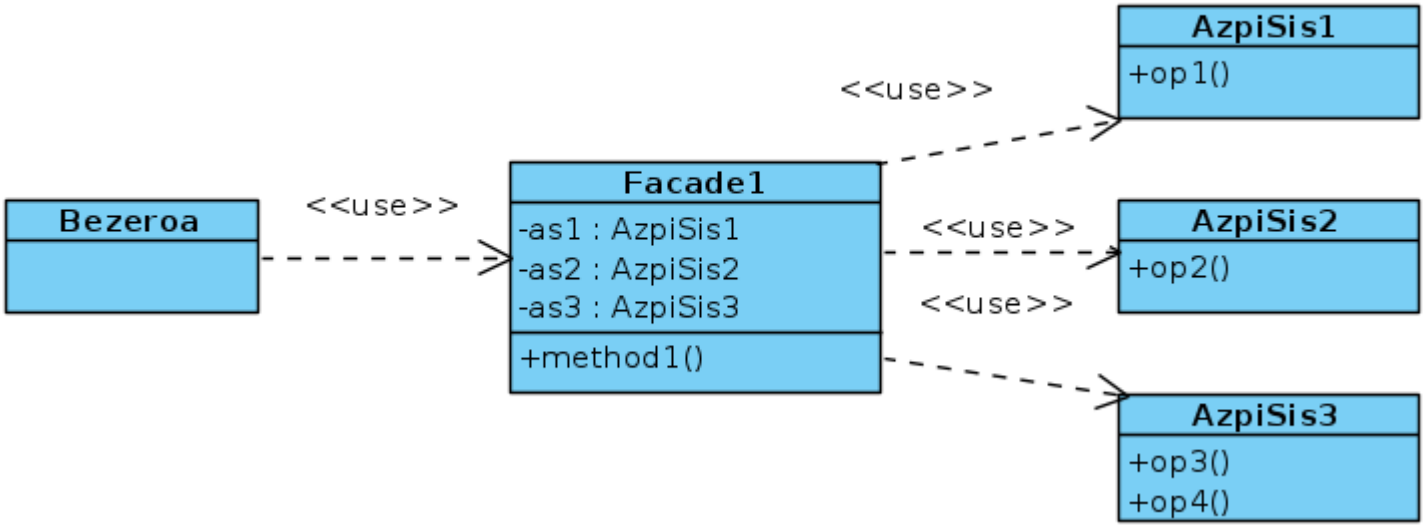
Klaseek beren artean duten **dependentzia maila** da. Zenbat eta akoplamendu txikiagoa, orduan eta eragin gutxiago izango dute sistemako aldaketek gure programan.

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

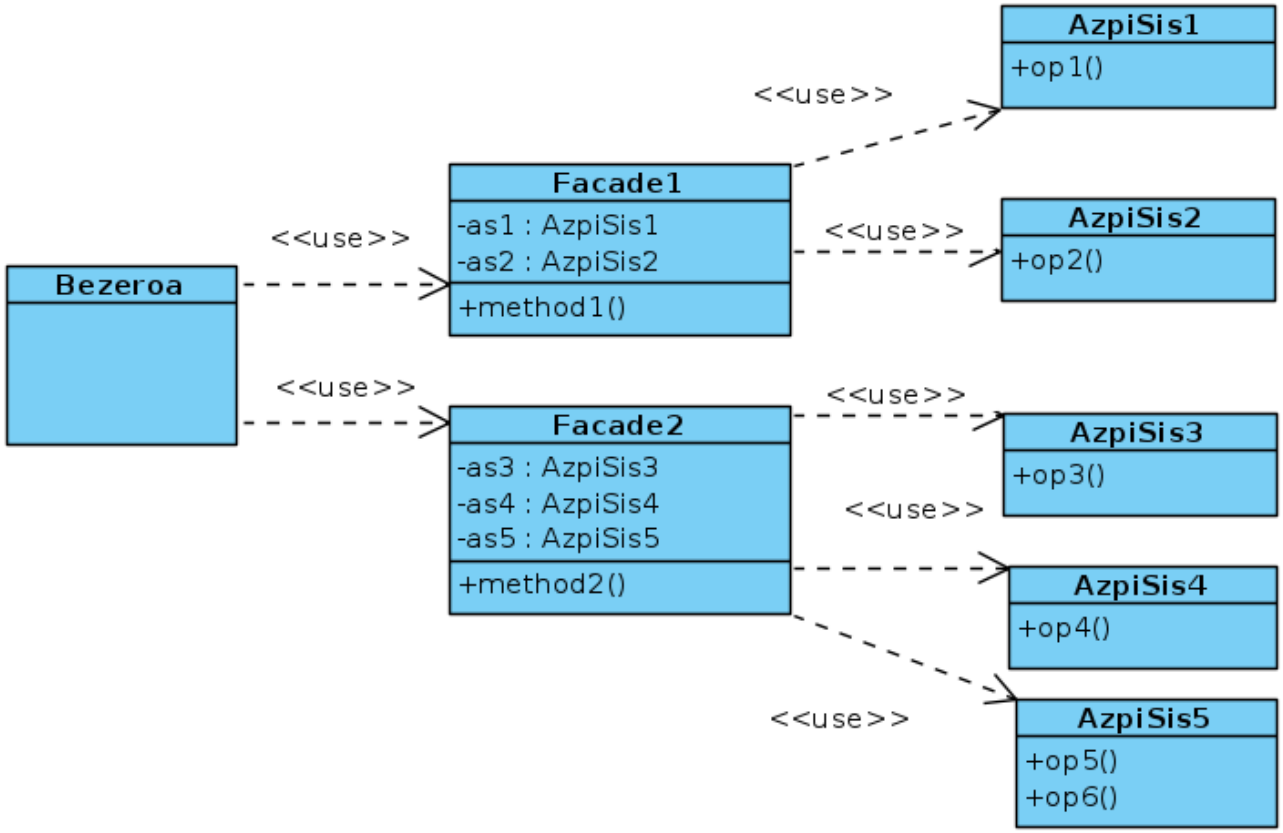
Facade: azpisistema multzo baten interfazeei interfaze bateratua ezarri; hau da, bezeroari maila altuko intefazea eskaini, azpisistemak erabilterrazagoak bihurtzeko



Ezaugarriak

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

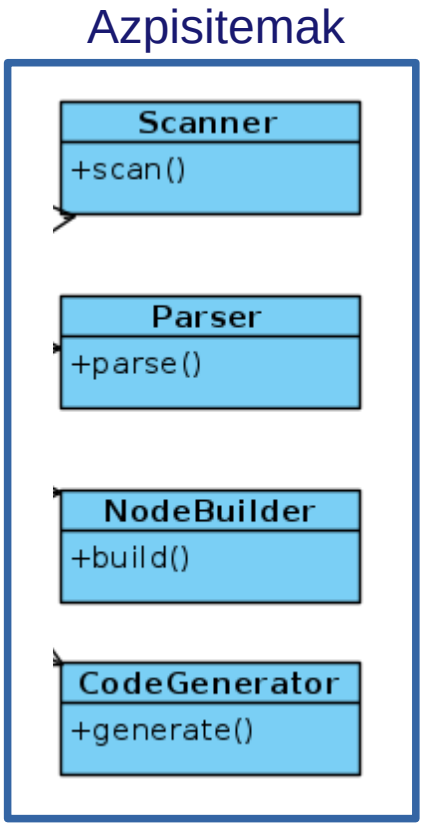
Orokortuz


AzpiSis1
+op1()
<<use>>
Facade1
-as1 : AzpiSis1
-as2 : AzpiSis2
+method1()
<<use>>
AzpiSis2
+op2()
<<use>>
Bezeroa
<<use>>
Facade2
-as3 : AzpiSis3
-as4 : AzpiSis4
-as5 : AzpiSis5
+method2()
<<use>>
AzpiSis3
+op3()
<<use>>
AzpiSis4
+op4()
<<use>>
AzpiSis5
+op5()
+op6()

Arazoa

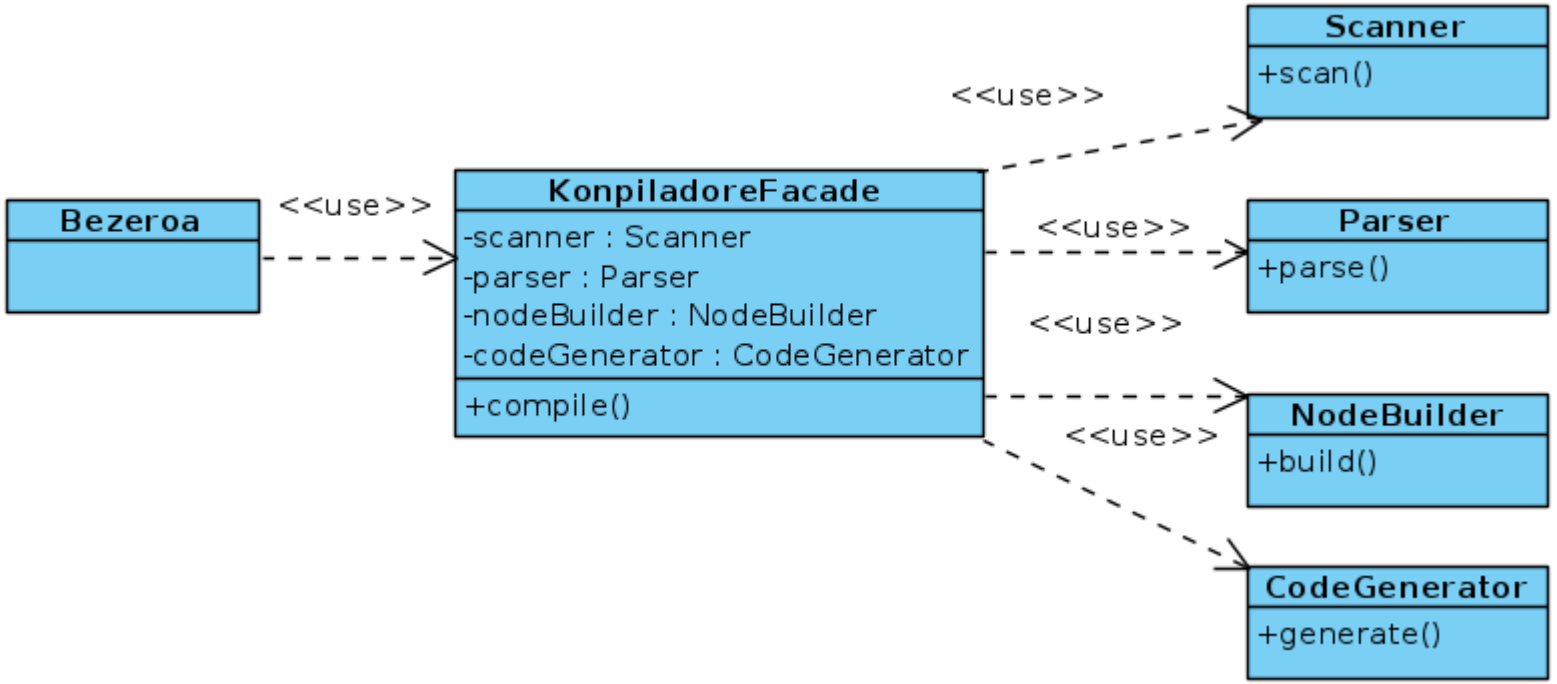
- Java konpiladore baten diseinua
- Konpilatzeko, azpisistema desberdinak:
  - *Scanner*-ak programa irakurri
  - *Parser*-ak prozesatu
  - *NodeBuilder*-ak zuhaitza sortu
  - *CodeGenerator*-ak bytecode-a sortu
- Konpiladorearen klase diagrama egin, gerora azpisistemak aldatuko direla kontutan izanik.

Ebazpena



* Sinplifikatze aldera, ez dira parametroak diagraman gehitu

Ebazpena



* Sinplifikatze aldera, ez dira parametroak diagraman gehitu

Ebazpena

```
public class KonpiladoreFacade {
    private static KonpiladoreFacade nKF;
    private Scanner scanner;
    private Parser parser;
    private NodeBuilder nodeBuilder;
    private CodeGenerator codeGenerator;

    private KonpiladoreFacade (){
        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();
    }
}
```

Azpisistema guztiak interfaze berean bateratu, erabilterrazagoa!!

Ariketa: Multimedia Gela

- Multimedia gela bat kudeatzeko sistema inplementatu.
- Bi motatako ekitaldiak:
  - Pelikula emanaldiak: **`pelikulaJarri`** metodoa
  - Hitzaldiak: **`aurkezpenaEgin`** metodoa

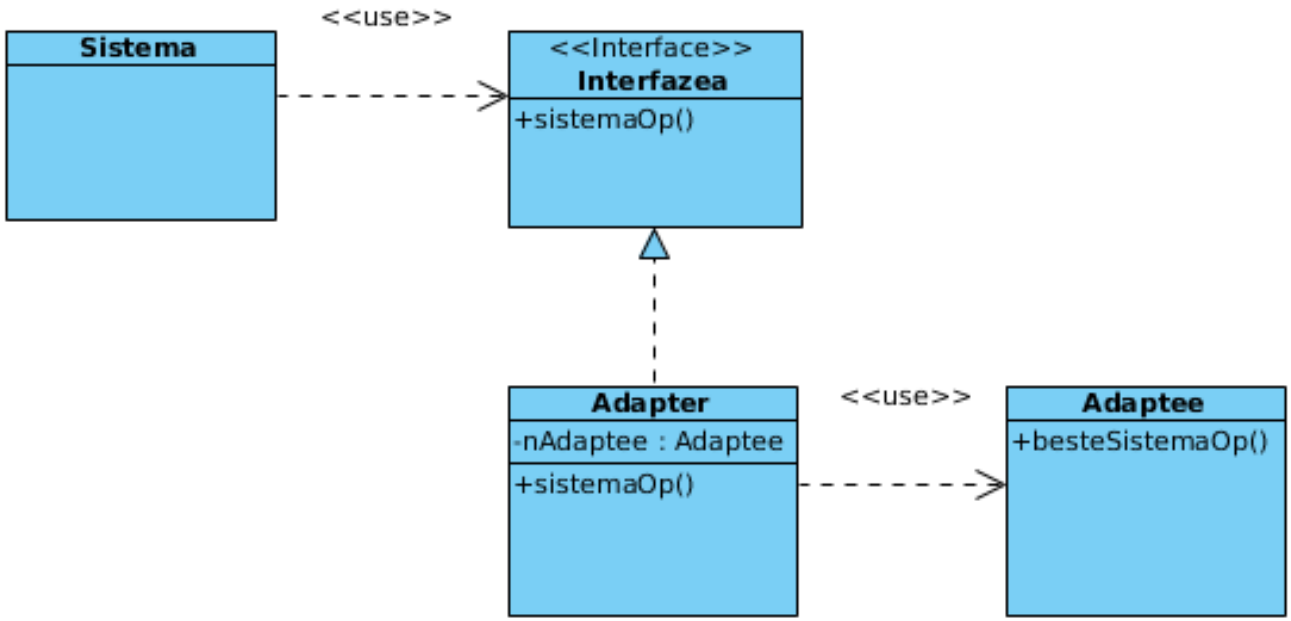
Ariketa: Multimedia Gela

- Pelikula emanaldietan: *pantaila jaitsi, proiektorea piztu, proiektorea DVD moduan jarri, DVD-a piztu, bozgorailuak piztu, bere bolumena finkatu, diskoa sartu* eta *diskoa martxan jarri*.
- Hitzaldietan: *pantaila jaitsi, proiektorea piztu, proiektorea PC moduan jarri, ordenagailua piztu* eta *aurkezpena martxan jarri.*
- Sistemaren klase diagrama eta ekitaldi mota bakoitza kudeatzeko zatiaren inplementazioa

Adapter

Eskema Orokorra

Adapter: interfaze bateraezinei elkarrekin lan egiteko aukera eman. Bezeroaren interfazeak gure sistemako funtzionalitate baliokidea dauka, baina, intefaze desberdinarekin; bitartekari lana egiten du.



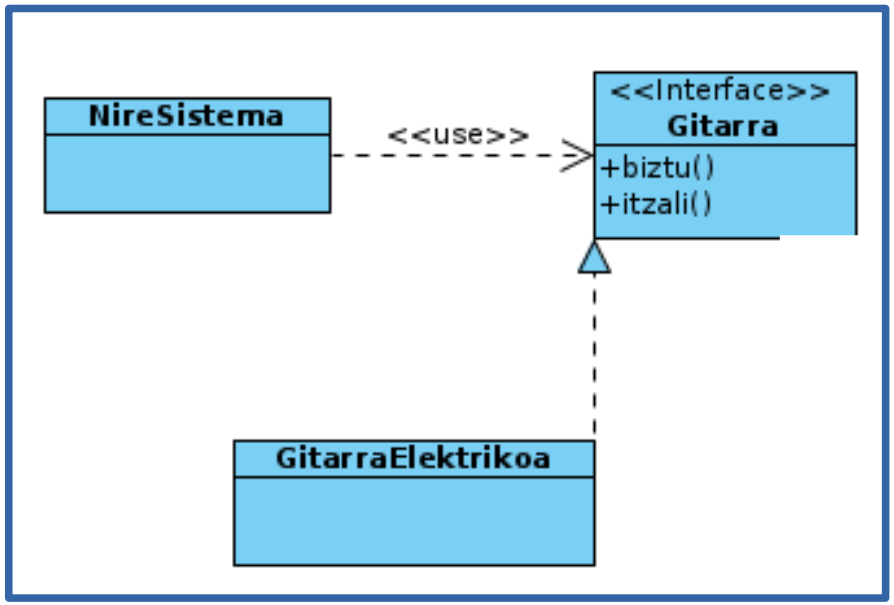
Ezaugarriak

- Berrerabilgarritasuna hobetu
- Hedapena erraztu
- Adaptee ez da ukitzen
  - Kodea agian ez dago eskuragarri

Arazoa

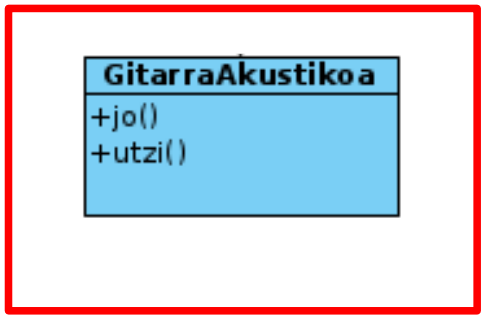
- Musika tresnak simulatzeko sistema:
  - Gitarrak simulatzeko *Gitarra* interfazea dugu, `piztu` eta `itzali` metodoekin.
  - Interfaze hori *GitarraElektriko* klaseak inplementatzen du.
- Beste sistema bateko *GitarraAkustiko* klasea berrerabiliko dugu, baina, `jo` eta `utzi` metodoak ditu.
- Sistemaren klase diagrama egin, berrerabilgarria izan behar duela kontutan hartuz.

Ebazpena

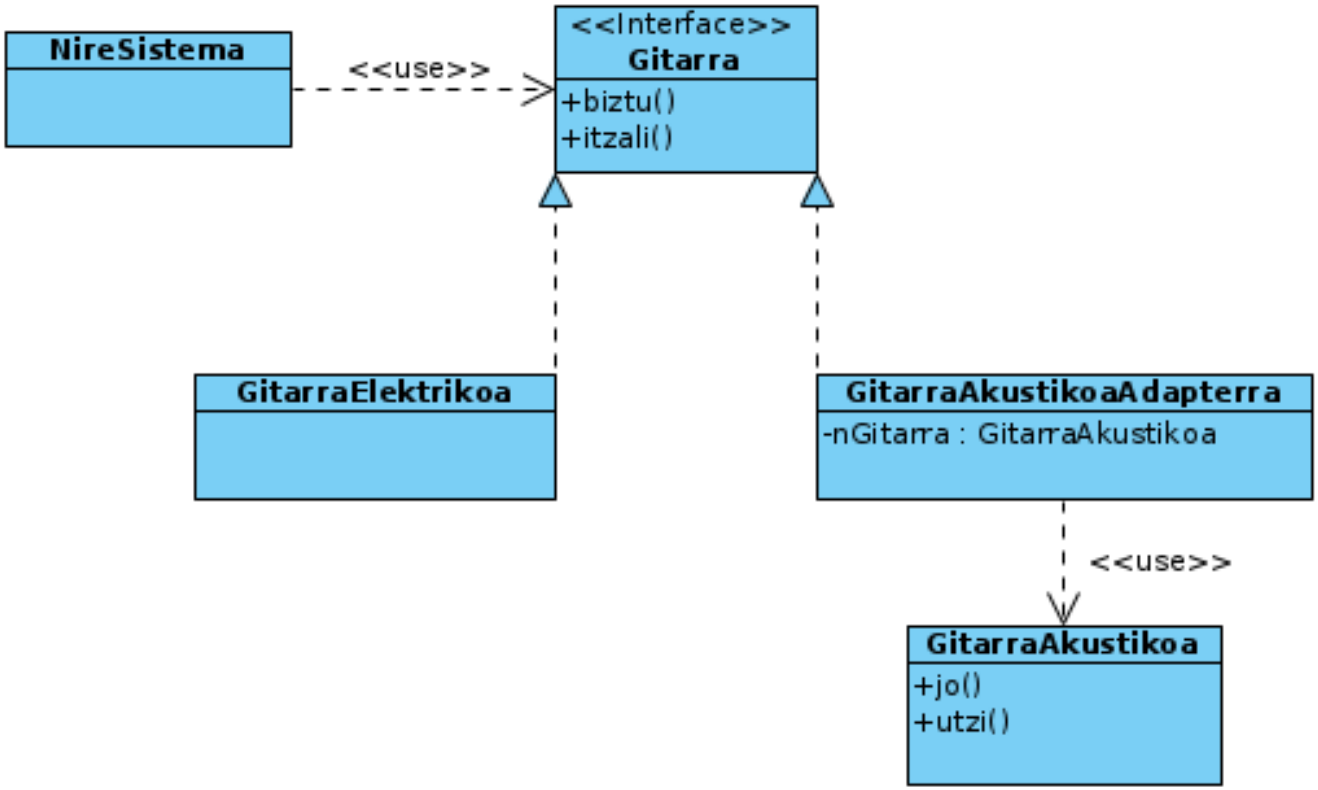


Jatorrizko sistema

Kanpoko sistema



Ebazpena


NireSistema
<<use>>
<<Interface>>
Gitarra
+biztu()
+itzali()
GitarraElektrikoa
GitarraAkustikoaAdapterra
-nGitarra : GitarraAkustikoa
<<use>>
GitarraAkustikoa
+jo()
+utzi()

Ebazpena

```
public interface Gitarra {
    public void piztu();
    public void itzali();
}
public class GitarraElektrikoa implements Gitarra {
    public void piztu(){…}
    public void itzali(){…}
}
```

Metodo baliokideak,
baina ez bateragarriak

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

Ebazpena

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

Metodo baliokideak, bateragarri egin

Ariketa: Motorrak

- Audi kotxeak kontrolatzeko sistema dugu
  - 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.

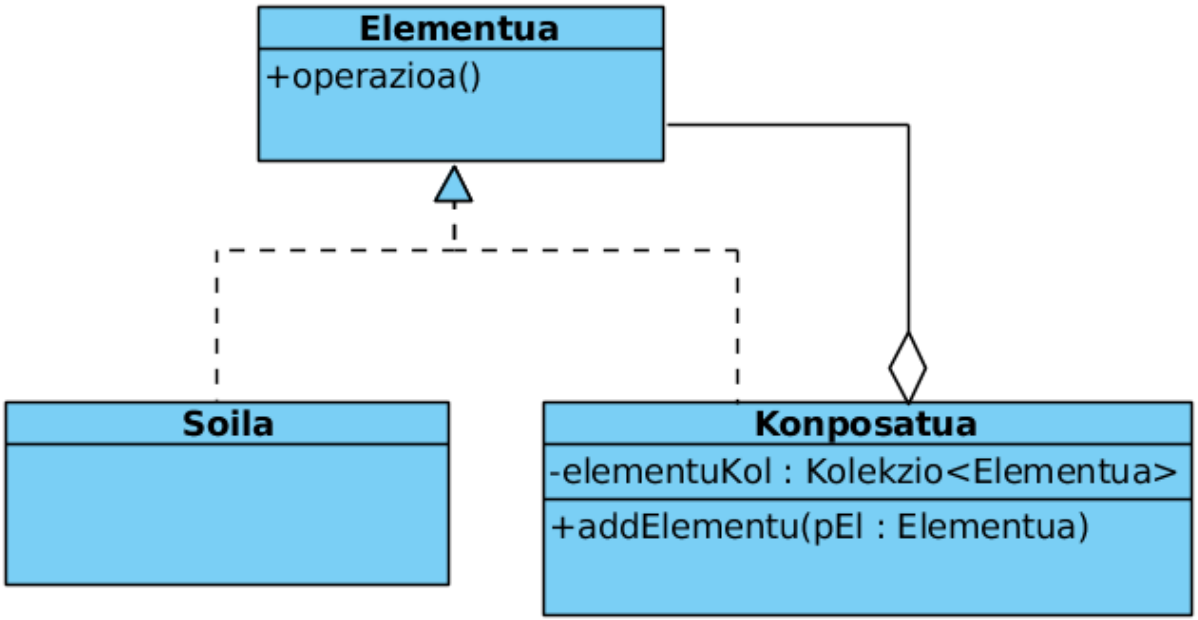
Ariketa: Motorrak

- Hurrengoa burutu ezazu:
  - Sistemaren diseinua (Klase Diagrama)
  - Gure sisteman motor elektrikoak sartzea ahalbidetuko duen zatiaren inplementazioa.

Composite

Eskema Orokorra

Composite: aplikazioan *banakako* objektuak eta *konposatuak* modu berean erabiltzea ahalbidetu. Zuhaitz motako hierakietan txertatu objektu horiek.



Ondorioak

- Banakako elementuak (*hostoak*) eta konposatuak (*nodoak*) zuhaitz egitura berean txertatu
  - *Part-whole* hierarkiak
  - Konposatuak elementu soilez zein konposatuez osatuta egon daitezke
- Objektu guztiek interfaze bera
  - *Nodo* zein *hosto*, era berean tratatu

Arazoa

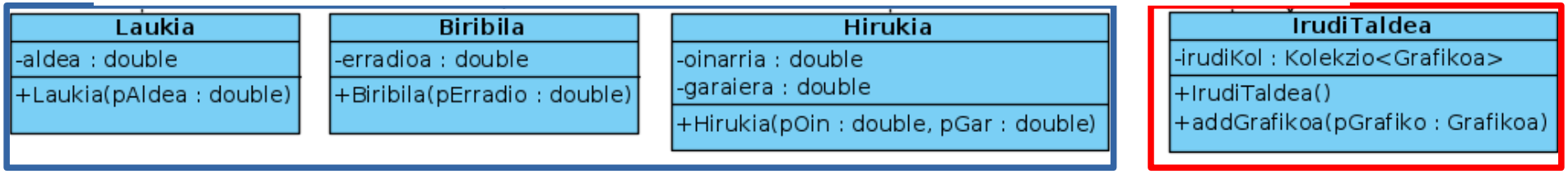
- Aplikazio baten elementu grafikoen informazioa biltzeko klaseak behar ditugu. Adibidez, *biribilak, laukiak* eta *hirukiak*.

- *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 (*irudi* zein *irudi multzo*) klase bat definitu daiteke, dagokion `marraztu()` metodoarekin.
- Baina, nola definitu irudi multzo bat irudi bakarra balitz bezala kudeatzeko?

Ebazpena

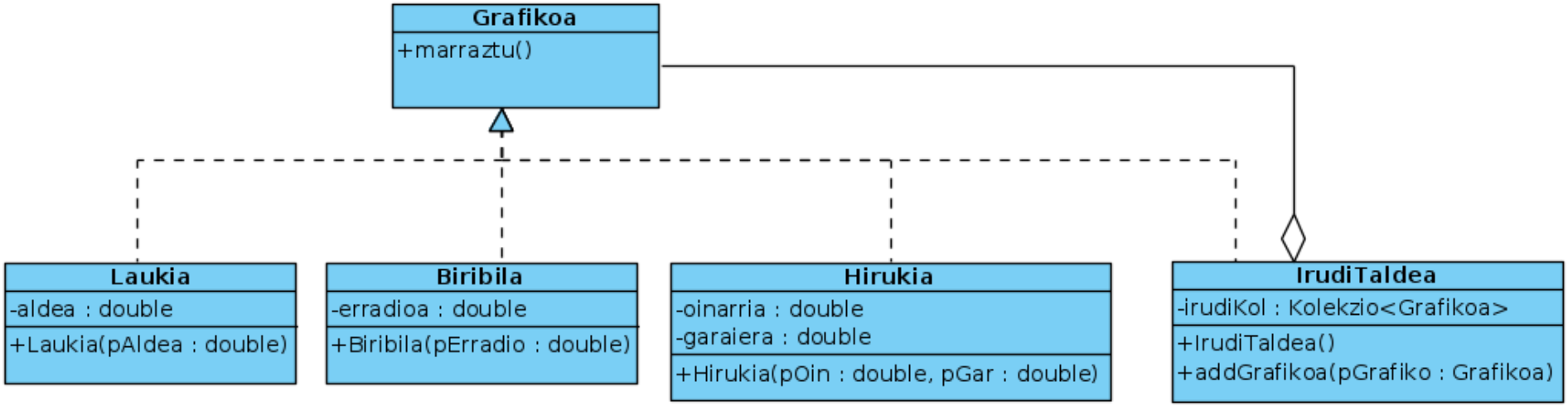


Hostoak

Nodoa

Irudi talde batek *Grafikoa* objektuen kolekzio bat du. Grafiko horiek *Biribilak*, *Laukiak, Hirukiak* edo beste irudi talde bat izan daitezke

Ebazpena



Irudi talde batek *Grafikoa* objektuen kolekzio bat du. Grafiko horiek *Biribilak*, *Laukiak, Hirukiak* edo beste irudi talde bat izan daitezke

Ebazpena

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

public interface Grafikoa{ public void marraztu();}

// ELEMENTU KONPOSATUA (NODOA)
class IrudiTaldea implements Grafikoa{
            private List<Grafikoa> irudiKol = new ArrayList<Grafikoa>();
           public void marraztu() {
                  for (Grafikoa grafikoa : irudiKol){
                        grafikoa.marraztu();
                   }
            }
            public void addGrafiko(Grafikoa grafikoa) {
                  irudiKol.addGrafikoa(grafikoa);
             }
  }
```

Denak berdin tratatu, hosto zein nodo.

Grafiko kolekzioa, hosto zein nodo

Grafiko guztiak marraztu, hosto zein nodo

Grafikoa gehitu, hosto zein nodo

Ebazpena

```
// ELEMENTU SOILA (HOSTOA)
class Hirukia implements Grafikoa {
  public void marraztu() {
      System.out.println(“Hirukia");
  }
}
```

Ebazpena

```
/** Bezeroa*/
public class Proba {

   public static void main(String[] args) {
       //HOSTOAK hasieratu
       Hirukia hirukia1 = new Hirukia();
       Hirukia hirukia2 = new Hirukia();
       Hirukia hirukia3 = new Hirukia();
       Hirukia hirukia4 = new Hirukia()
       //NODOAK hasieratu
       IrudiTaldea talde1 = new IrudiTaldea();
       IrudiTaldea talde2 = new IrudiTaldea ();
       IrudiTaldea talde3 = new IrudiTaldea ();
```

Ebazpena

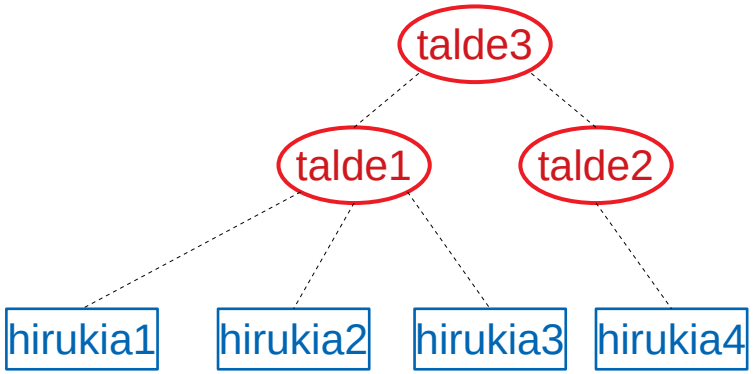
```
        //NODOAK osatu
        talde1.add(hirukia1);
        talde1.add(hirukia2);
        talde1.add(hirukia3);
        talde2.add(hirukia4);
        talde3.add(talde1);
        talde3.add(talde2);

        talde3.marraztu();
    }
}
```

1. nodoa

2. nodoa

3. nodoa

Errekurtsiboki, “Hiruki” guztiak marraztu



Ariketa

- Laborategian ikusitako *SWING* liburutegiko osagai eta edukiontzien egitura bat sortuko dugu. Demagun hurrengoak ditugula soilik:
  - Konposatuak: *JFrame* eta *JPanel*
  - Soilak: *JButton* eta *JLabel*
- Osagai orok `mugitu` eta `tamainaAldatu` metodoak izango ditu.

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