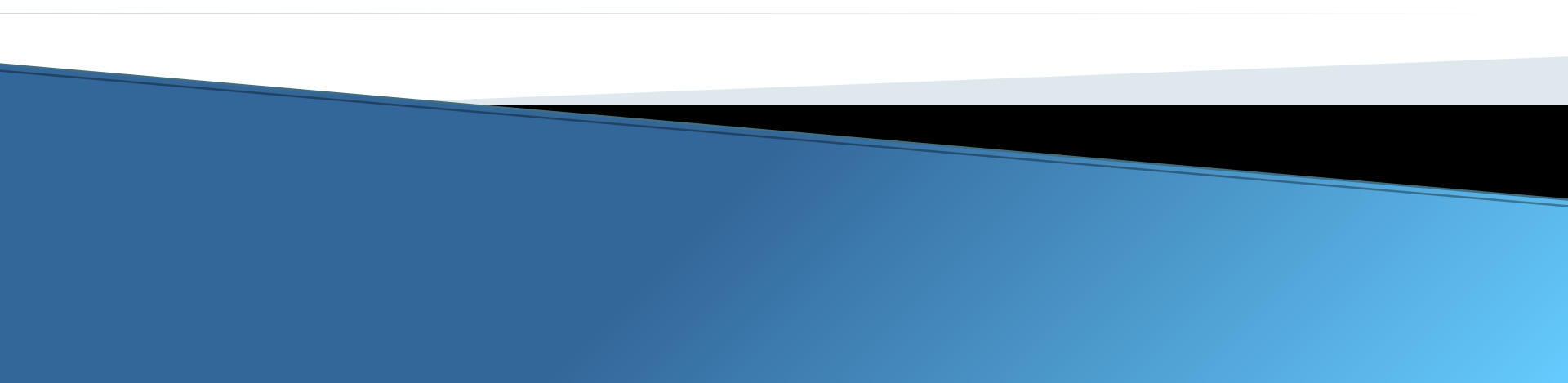


SOLID Printzipioak eta Diseinu Patroiak



Sarrera - Aldaketa

“Software sistemek bizi zikloan zehar aldaketak jasaten dituzte”

- Diseinu onei eta txarrei gertatzen zaie
- Diseinu onak egonkorak dira

Berdin da non zauden lanean, zer eraikitzen ari zaren edo zein programazio lengoaia erabiltzen ari zaren, beti egongo da konstante bat, **aldaketa**.

Zure aplikazioa oso ondo diseinatzen baduzu ere, denbora pasa ala aplikazioa hazi egin beharko da edo **aldaketak** jasan beharko ditu, bestela zaharkituta geldituko da.

Konstantea den gauza bakarra **aldaketa** da.

SOLID Printzipioak

- **S**ingle-Responsability Principle (SRP)
- **O**pen-Close Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

Irakurketa:

<http://mundogeek.net/archivos/2011/06/09/principios-solid-de-l-a-orientacion-a-objetos/>

Single Responsibility Principle (SRP)



Single Responsibility Principle

Just because you *can* doesn't mean you *should*.

Irudiak hemenetik hartuta daude:

<http://blogs.msdn.com/b/cdndevs/archive/2009/07/15/the-solid-principles-explained-with-motivational-posters.aspx>

SRP

Moduluek eta klaseek SWeko **funtzionalitate bakarraren ardura** izan behar dute

SRP Adibidea

```
public class CurrencyConverter {  
  
    public BigDecimal convert(Currency from, Currency to, BigDecimal amount) {  
        // gets connection to some online service and asks it to convert currency  
        // parses the answer and returns results  
    }  
  
    public BigDecimal getInflationIndex(Currency currency, Date from, Date to) {  
        // gets connection to some online service to get data about  
        // currency inflation for specified period  
    }  
}
```

Zergatik kalkulatzeko
da inflazioa txanpon
trukaketarekin?

Eta txanpon trukaketa
zerbitzua aldatuko
balitz? Edo inflazioa
kalkulatzeko formatua?

**Ez da intuitiboa!
Gainkargatuta dago!**

**Klasea bi kasuetan
aldatu behar da!**

Konpondu!!!

SRP Adibidea

```
public class CurrencyConverter {  
    public BigDecimal convert(Currency from, Currency to, BigDecimal amount) {  
        // gets connection to some online service and asks it to convert currency  
        // parses the answer and returns results  
    }  
}
```

Inflazioa kalkulatzeko
formatua aldatzen bada?
InflationIndexCounter
bakarrik aldatzen dugu!

Txanpon trukaketa
zerbitzua aldatzen bada?
CurrencyConverter
bakarrik aldatzen dugu!

```
public class InflationIndexCounter {  
    public BigDecimal getInflationIndex(Currency currency, Date from, Date to) {  
        // gets connection to some online service to get data about  
        // currency inflation for specified period  
    }  
}
```

SRP Adibidea II

Bi ardura:
Kautotzea eta
erabiltzailea DBtik
lortzea

```
public class UserAuthenticator {  
    public boolean authenticate(String username, String password){  
        User user = getUser(username);  
        return user.getPassword().equals(password);  
    }  
}
```

```
private User getUser(String username){  
    st.executeQuery("select user.name, user.password from user where id  
    // something's here  
    return user;  
}
```


SRP Adibidea II

```
public class UserAuthenticator {  
    private UserDetailsService userDetailsService;  
    public UserAuthenticator(UserDetailsService service) {  
        userDetailsService = service;  
    }  
    public boolean authenticate(String username, String password){  
        User user = userDetailsService.getUser(username);  
        return user.getPassword().equals(password);  
    }  
}
```



Orain ez dugu zuzenean
DBarekin lan egiten



Kautoketa LDAP-era aldatuta,
Klasea ez da aldatzen

Open-Close Principle (OCP)



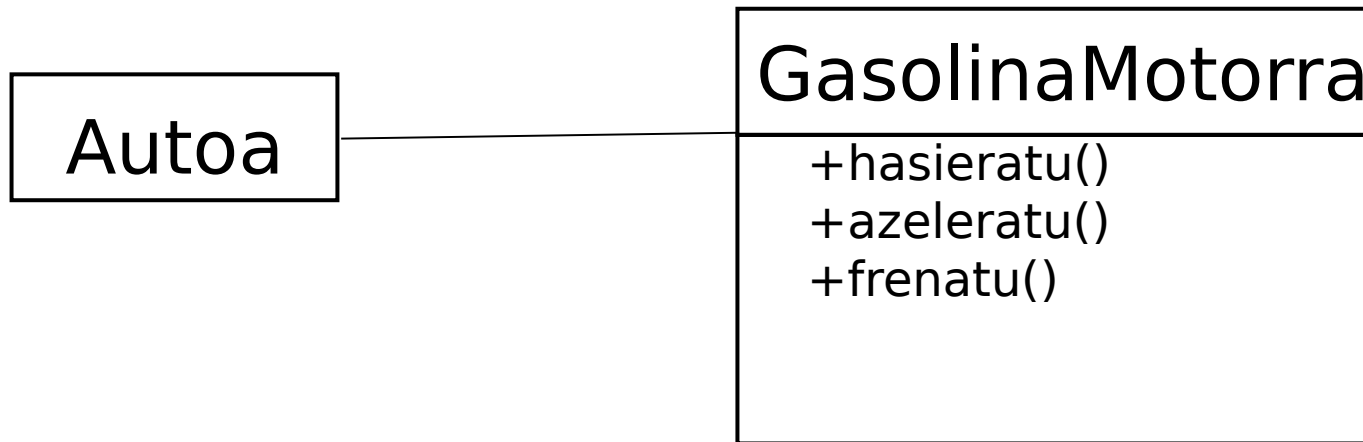
Open-Closed Principle

Open-chest surgery isn't needed when putting on a coat.

OCP

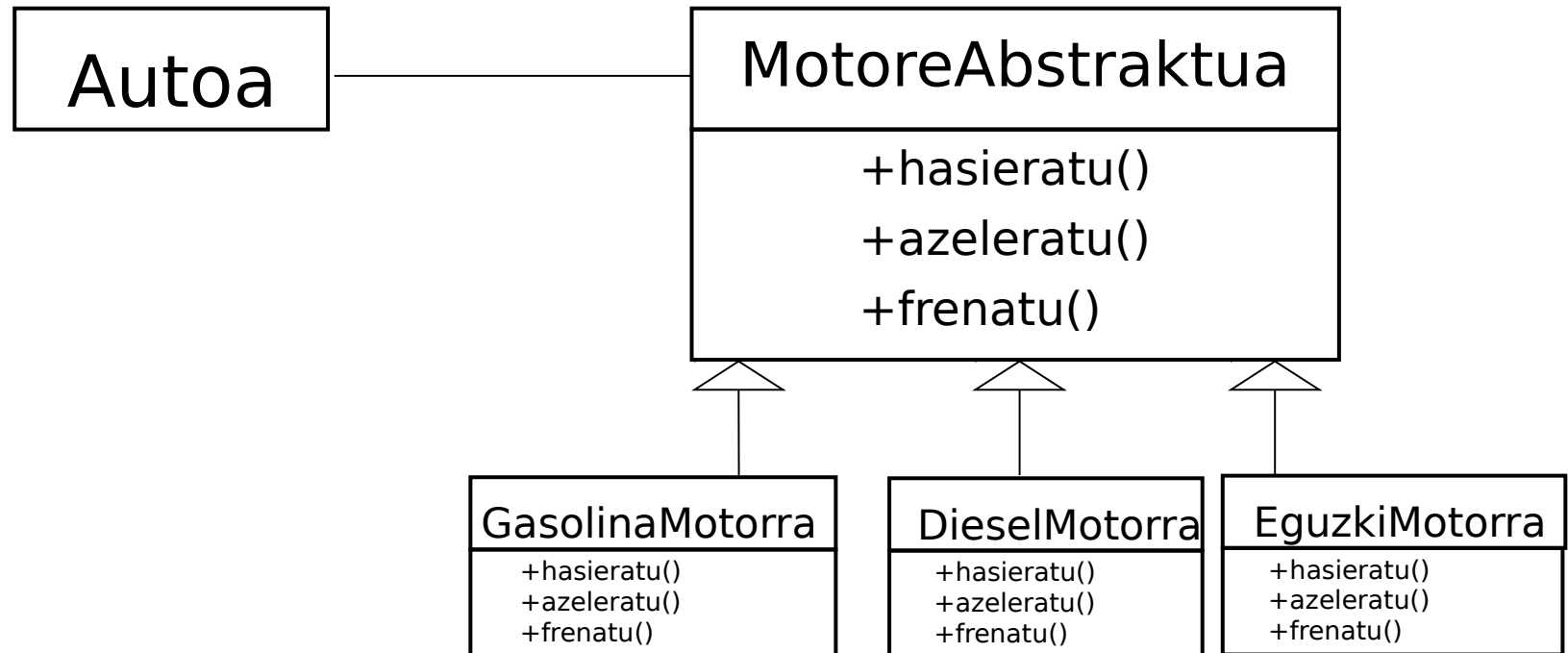
SW entitateak (funtzionalitateen) **hedapenerako irekiak**, baina (kodearen) **aldaketarako itxiak** izan behar dute

OCP Adibidea



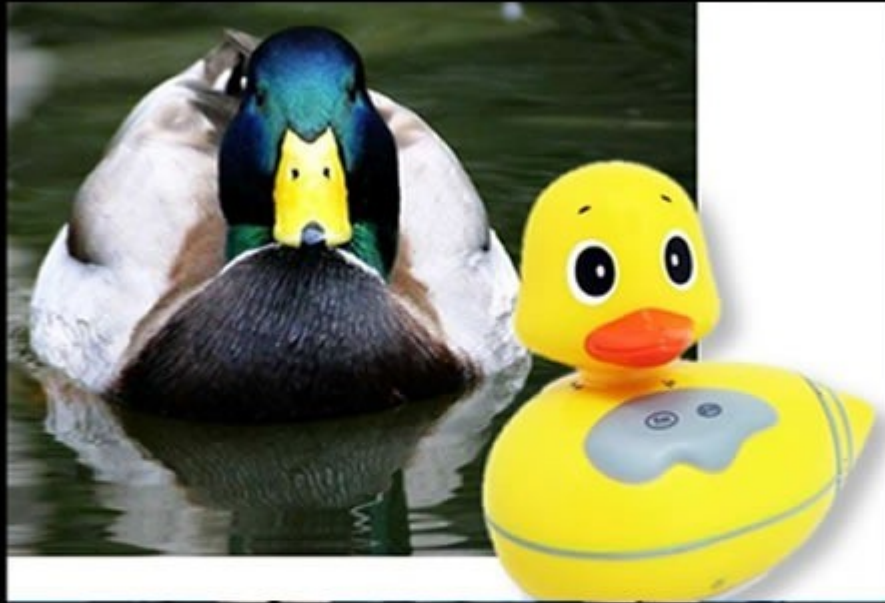
- Nola egin Auto batek *GasolinaMotorra* edo *EguzkiMotorra* erabili ditzan?
- Autoa klasea aldatu behar dugu!
 - ...gutxienez diseinu honetan

OCP Adibidea



- Ahal den elean, klase batek ez du klase konkretu batekin dependentziarik izan behar
- Klase abstraktu batekin izan behar du dependentzia.

Liskov Substitution Principle (LSP)



Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries, you probably have the wrong abstraction.

LSP

Klase batetik heredatzen duen azpiklase
(**seme klase**) **orok** lehenegoa (**ama klasea**) **bezala**
erabili ahal beharko litzateke, beren arteko
desberdintasunak ezagutu barik ere

LSP herentzian oinarritzen da

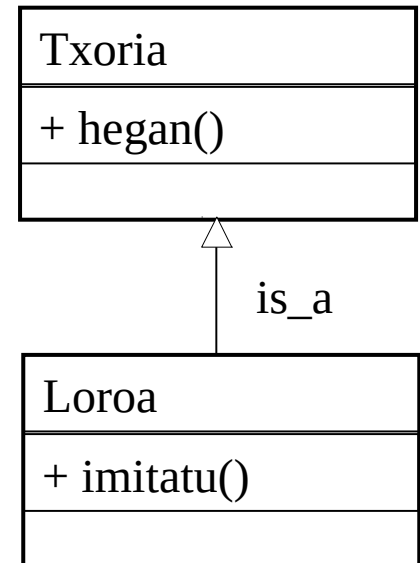
*Herentziak hurrengoa bermatu behar du: **superklasearen edozein objekturen propietate frogagarri azpiklaseen edozein objekturentzat baliogarria da.***

B. Liskov, 1987

Herentziak erraza dirudi...

```
abstract class Txoria { // lumak, hegoak... ditu
    public void hegan(); // Txoriek hegan egin dezakete
};

class Loroa extends Txoria { // Loro bat txori bat da
    public void imitatu();      // Hitzak errepikatu ditzake
};
// ...
Loroa nireMaskota=new Loroa();
nireMaskota.imitatu();        // Loroa izanda, imitatu() dezake
nireMaskota.hegan();          // Txoria izanda hegan() egin dezake
```



Pinguinoek ezin dute hegan egin

```
class Pinguinoa extends Txoria {  
    public void hegan() {  
        new Exception("ezin dut hegan egin!"); }  
};
```



```
void txoriakBezalaHegan (Txoria txori) {  
    txori.hegan(); // lora ondo  
    // Eta pinguinoa?...OOOPS!!  
}
```

- Ez du “*Pinguinoek ezin dute hegan egin*” modelatzen
- “*Pinguinoek hegan egin dezakete, baina saiatuz gero errorea*” modelatzen du
- Hegan egiten saiatzen badira → Run-time errorea
- Ordezkapen printzipioan pentsatu → Liskov printzipioa ez da betetzen

Interface Segregation Principle (ISP)



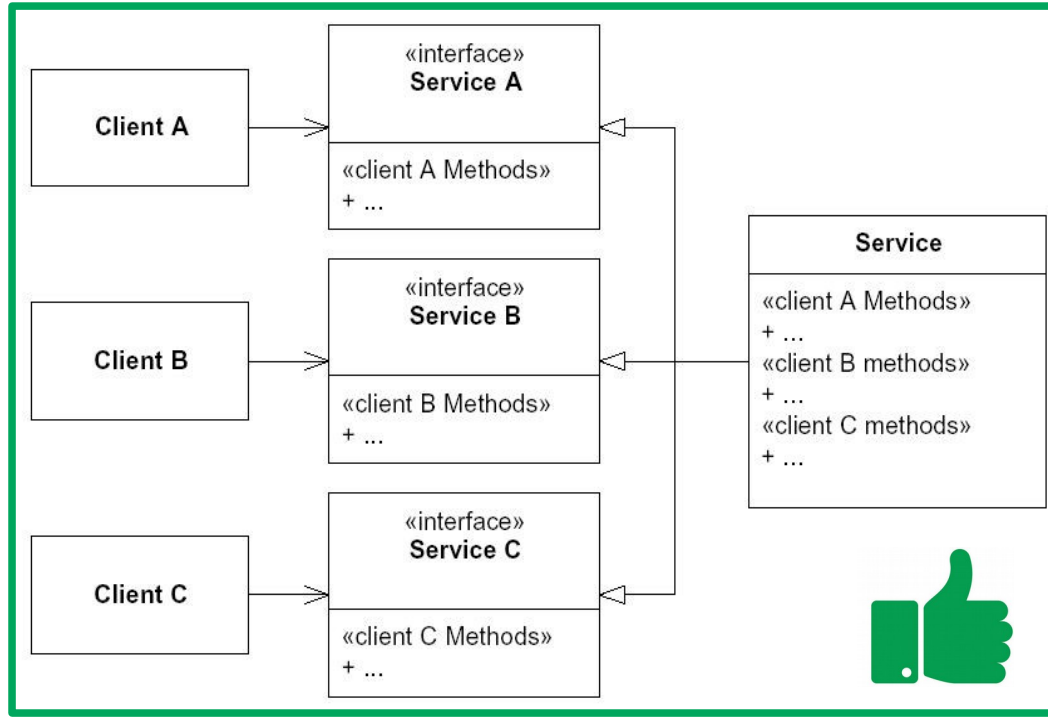
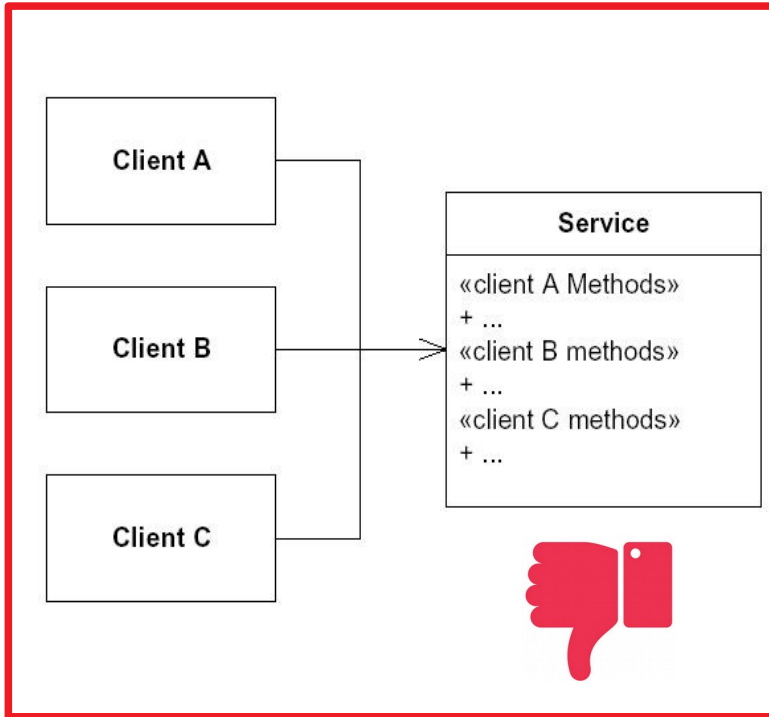
Interface Segregation Principle

You want me to plug this in *where?*

ISP

**Bezeroek, erabiltzen ez duten metodoen
dependentziarik ez dute izan behar**

Adibidea



Dependency Inversion Principle (DIP)



Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

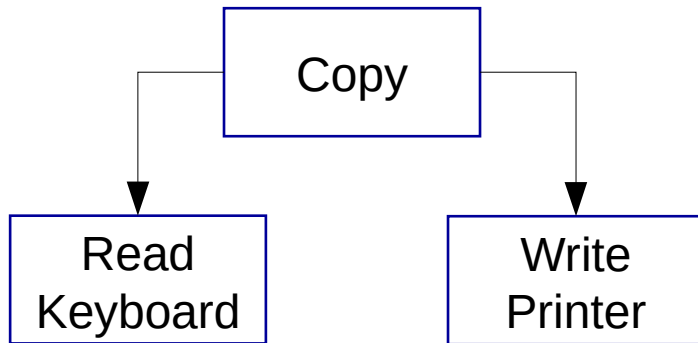
DIP

- I. **Goi mailako moduluek** ezin dute behe mailako moduluekin menpekotasunik izan. Goikoek zein behekoek, **menpekotasuna abstrakzioekin**.
- II. Abstrakzioek ezin dute xehetasunekin menpekotasunik izan. **Xehetasunek abstrakzioekin menpekotasuna**.

Martin, 1996

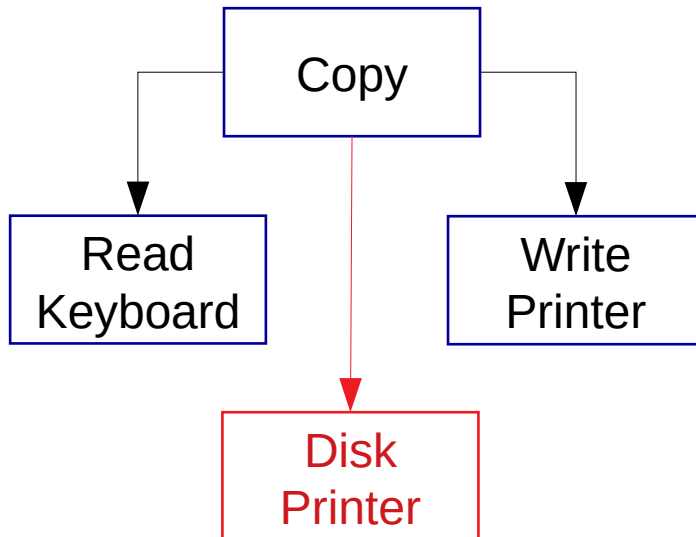
- OCPk helburua adierazten du. DIPek mekanismoa.
- Superklase batek ez ditu bere azpiklaseak ezagutu behar.
- Inplementazio xehetasunak dituzten moduluek ez dute beraien arteko menpekotasunik. Menpekotasuna abstrakzioen bidez definitzen dira.

Adibidea



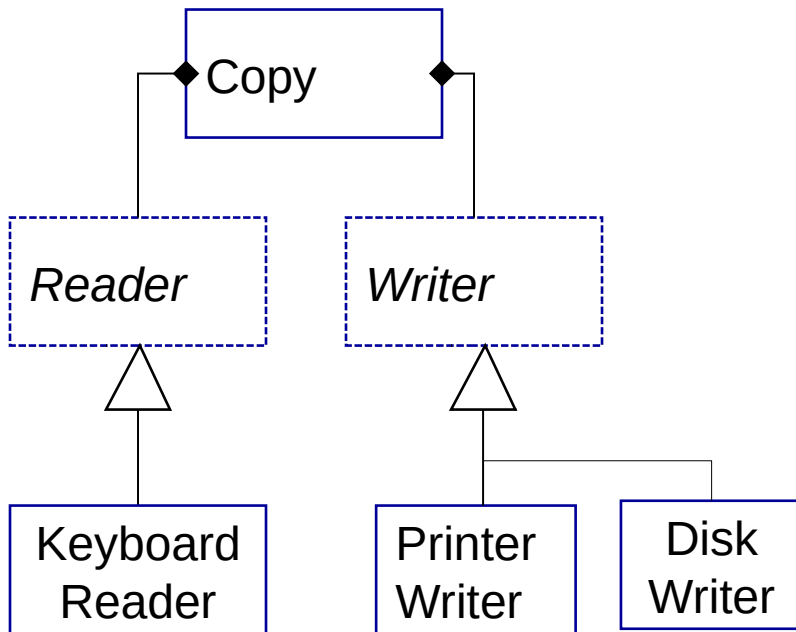
```
void Copy(){  
    int c;  
    while ((c = ReadKeyboard()) != EOF)  
        WritePrinter(c);  
}
```


Adibidea



```
enum OutputDevice {printer, disk};  
void Copy(OutputDevice dev){  
    int c;  
    while((c = ReadKeyboard())!= EOF)  
        if(dev == printer)  
            WritePrinter(c);  
        else  
            WriteDisk(c);  
}
```

Adibidea

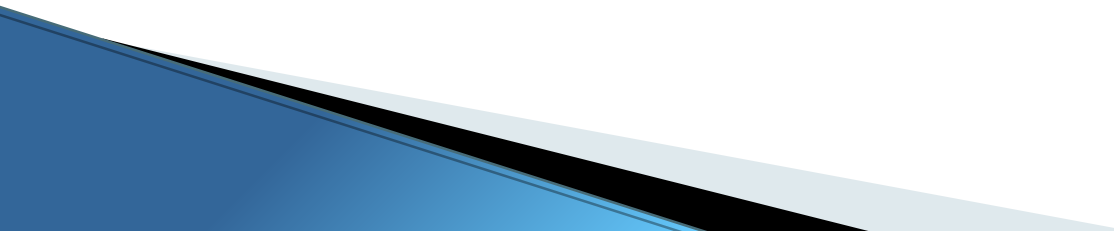


```
class Reader {
    public abstract int read(){ };
}

class Writer {
    public abstract void write(int i);
};

void copy(Reader r, Writer w){
    int c;
    while((c = r.read()) != EOF)
        w.write(c);
}
```

Laburbilduz

- ▶ SW sistemek **aldaketak** dituzte beren bizi zikloan
 - ▶ SW diseinuak aldaketetara **modatu** behar dira
 - ▶ SOLID printzipioak hastapeneko pausuak dira, diseinu konplexuak egiteko teknika sofistikatuagoak
- 

DISEINU PATROIAK

DISEINU PATROIAK

MOTIBAZIOA

- ▶ OZ diseinua, zaila!! Berrerabilgarritasuna, are zailago!!
- ▶ Iraganean funtzionatu duten soluzioak berrerabili
- ▶ PATROIEK diseinu problema zehatzak ebatzi
 - Diseinu malgua eta berrerabilgarria ahalbidetu
- ▶ SOLID printzipioetan oinarritu

DISEINU PATROIAK

DEFINIZIOA: elkarrekin komunikatzen diren objektu eta klaseen definizioa, baina, diseinu problema orokor bat testuinguru partikular batetan ebazteko egokituta

DISEINU PATROIAK

HISTORIA

- ▶ 1964-1979: Christopher Alexanderrek patroiak planteatu *arkitektura munduan*.
- ▶ 1990-1992: “Gang of Four”(GOF) taldearen lana hasi, informatikara Alexanderren ideiak.
- ▶ 1995: “***Design Patterns, Elements of Reusable Object-Oriented Software***” liburu ospetsua argitaratu

DISEINU PATROIAK

SAILKAPENA

- ▶ **Sortzaileak:** objektuen sorkuntza
- ▶ **Egiturazkoak:** klase eta objektuen konposaketa
- ▶ **Portaerazkoak:** klase eta objektuen elkarreragin eta ardurak banatzeko era

PATROIAK: ITERATOR

MOTIBAZIOA

- ▶ Objektu sortak sekuentzialki zeharkatzeko metodoak asko erabili.
- ▶ Objektu sortak era ezberdinetan implementatu daitezke.

| Arrayetan oinarritutako zerrenda | Zerrenda estekatuetan oinarritutako zerrenda |
|---|--|
| <pre>public class ArrayZerrenda { private int osagaiKop; private String[] zerren; ... }</pre> | <pre>public class Nodoa { private String datua; private Nodoa hurr; ... } public class ZerrendaEstekatua{ private Nodoa lehena; ... }</pre> |

PATROIAK: ITERATOR

MOTIBAZIOA

- Inplementazio bakoitzerako, osagaiak zeharkatzeko metodo desberdina.

Arrayetan oinarritutako zerrenda

```
...  
String datua = null;  
for (int i= 0; i< zerren.size();i++) {  
    datua = zerren.get(i);  
    ...  
}  
...
```

Zerrenda estekatueta oinarritutako zerrenda

```
...  
String datua = null;  
Nodoa aux = zerren.getFirst();  
while (aux!=null) {  
    datua = aux.getContent();  
    ...  
    aux = aux.getNext();  
}  
...
```

PATROIAK: ITERATOR

ARAZOA

- ▶ Gauza bera egiteko, inplementazio desberdinak

HELBURUA

- ▶ Kolekzio bat sekuentzialki zeharkatzeko modu bat lortu, baina, bere adierazpenaren modu independentean

PATROIAK: ITERATOR

IMPLEMENTAZIOA

| Arrayetan oinarritutako zerrenda | Zerrenda estekatuetan oinarritutako zerrenda |
|--|--|
| <pre>ArrayList<String> lista = new ArrayList<String>(); ... String cadena = null; Iterator<String> iter = lista.iterator(); while (iter.hasNext()){ cadena = iter.next(); ... } ...</pre> | <pre>LinkedList<String> lista = new LinkedList<String>(); ... String cadena = null; Iterator<String> iter = lista.iterator(); while (iter.hasNext()){ cadena = iter.next(); ... } ...</pre> |

Ez da kolekzio konkretuaren egitura ezagutu behar osagaiak zeharkatu ahal izateko

PATROIAK: SINGLETON

MOTIBAZIOA

- ▶ Elementu bakarra puntu desberdinetik erreferentziatu behar denean, elementu hori beti berdina dela ziurtatu behar da. Instantzia bakarra egon behar da, beste guztiek objektu bera ikusteko.

PATROIAK: SINGLETON

DESKRIBAPENA

- ▶ Klaseak berak sortu bere instantzia bakarra.
- ▶ Sarrera globala klase-metodo batekin (estatikoa)
- ▶ Klaseko konstruktorea pribatua, instantzia berriak egitea ezinezkoa izateko.

PATROIAK: SINGLETON

IMPLEMENTAZIOA

```
private static Singleton instantziaBakarra = null;

// Konstruktore pribatua, beste klaseek instantzia berriak sortu ezin
private Singleton() {}

//Instantzia bakarra itzuli. Sortuta ez badago, sortu egiten du
public static Singleton getInstance() {
    if (instantziaBakarra == null) {
        instantziaBakarra = new Singleton();
    }
    return instantziaBakarra;
}
```