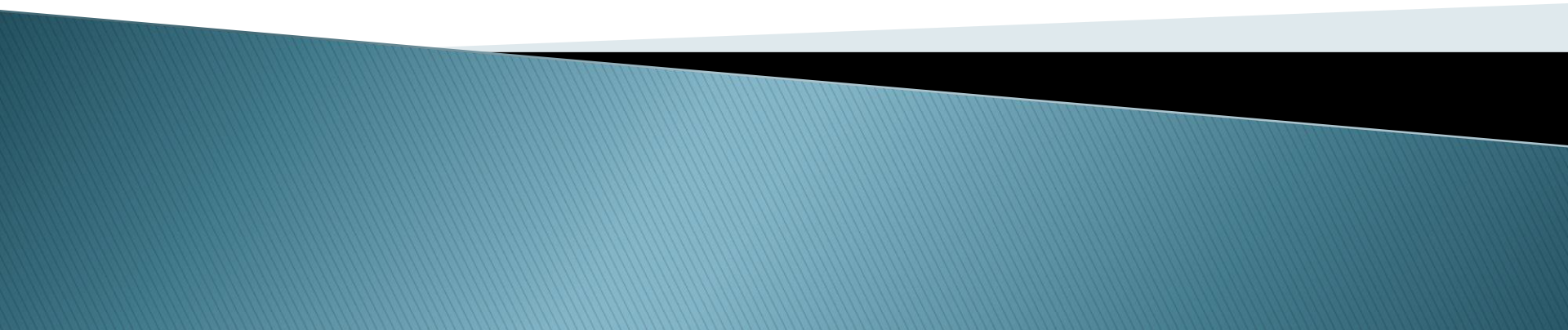


# SOFTWARE BERRERABILPENA

SOFTWARE INGENIARITZA



# EDUKIAK

- ▶ Motibazioa
- ▶ Definizioa
- ▶ Orokortzearen dimentsioak
  - Dimentsio bertikala
  - Dimensión horizontala
- ▶ Software berrerabilpena Java 8-n

# MOTIBAZIOA

- ◉ Askotan oso antzerakoak diren inplementazioak errepikatzen dira soilik datu ezberdinak erabiltzen direlako.
- ◉ Mundu errealeko adibide bat:  
Bideo, DVD edo telebistaren urrutiko agintea hondatu bada: **Zer egin dezakegu?**

a) Konpontzen saiatu

Arriskutsua eta neketsua

b) Urrutiko aginte berria fabrikatu

Arriskutsua eta neketsua

c) Dendan berri bat erosi

Garestia

d) Aginte unibertsala erosi (generikoa)

Merkea eta segurua

# MOTIBAZIOA

- ◉ Urrutiko agintea erostea erabakitzen badugu:

- Agintea gailu konkreturako konfiguratu:

Kode baten bitartez zein bideo, DVD edo telebistarekin egingo duen lan adierazten diogu



Funtzionalitatea moldatzen da

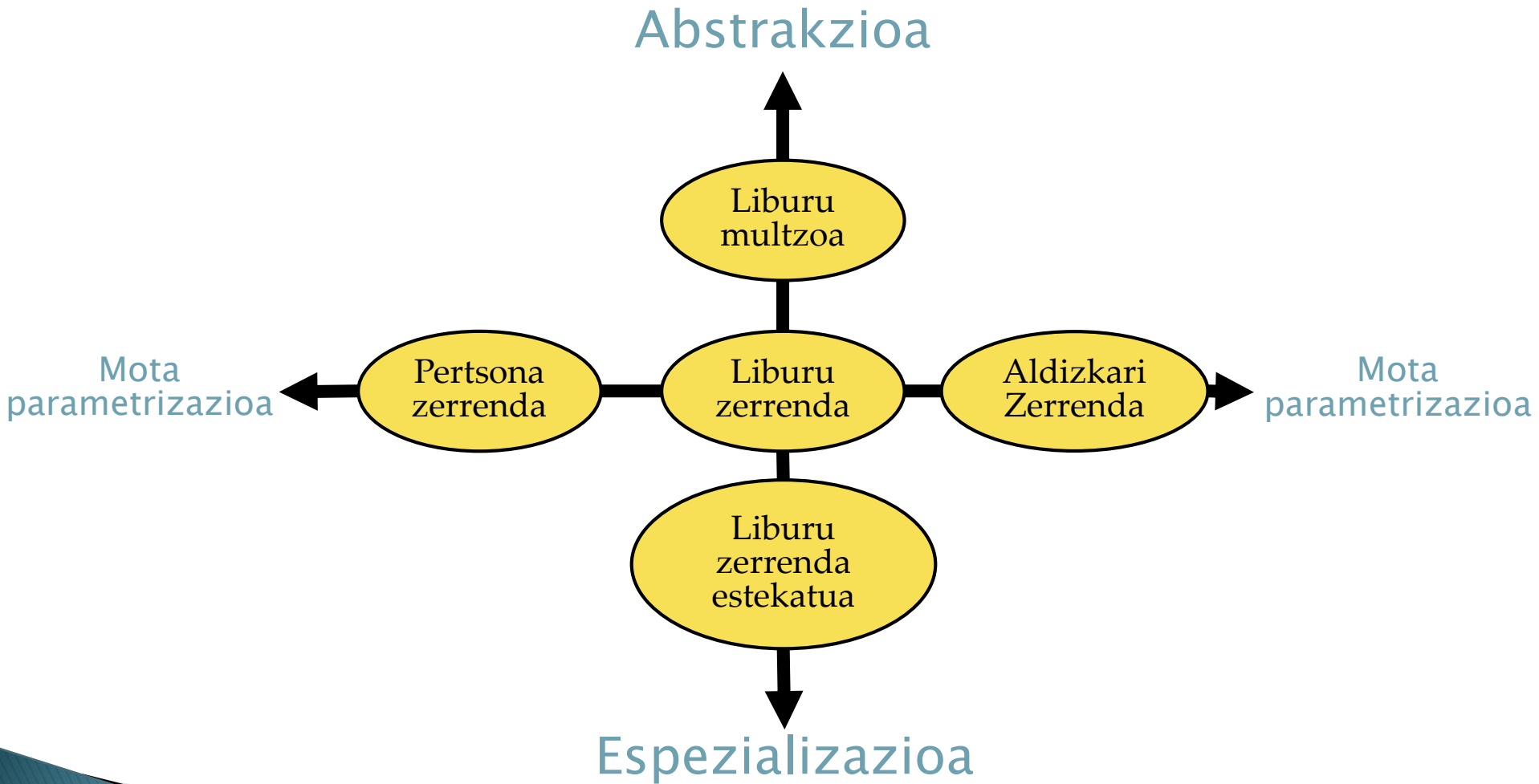
- Agintea originala balitz bezala erabiltzen da

*Zergatik ez erabili ideia hau software garapenean?*

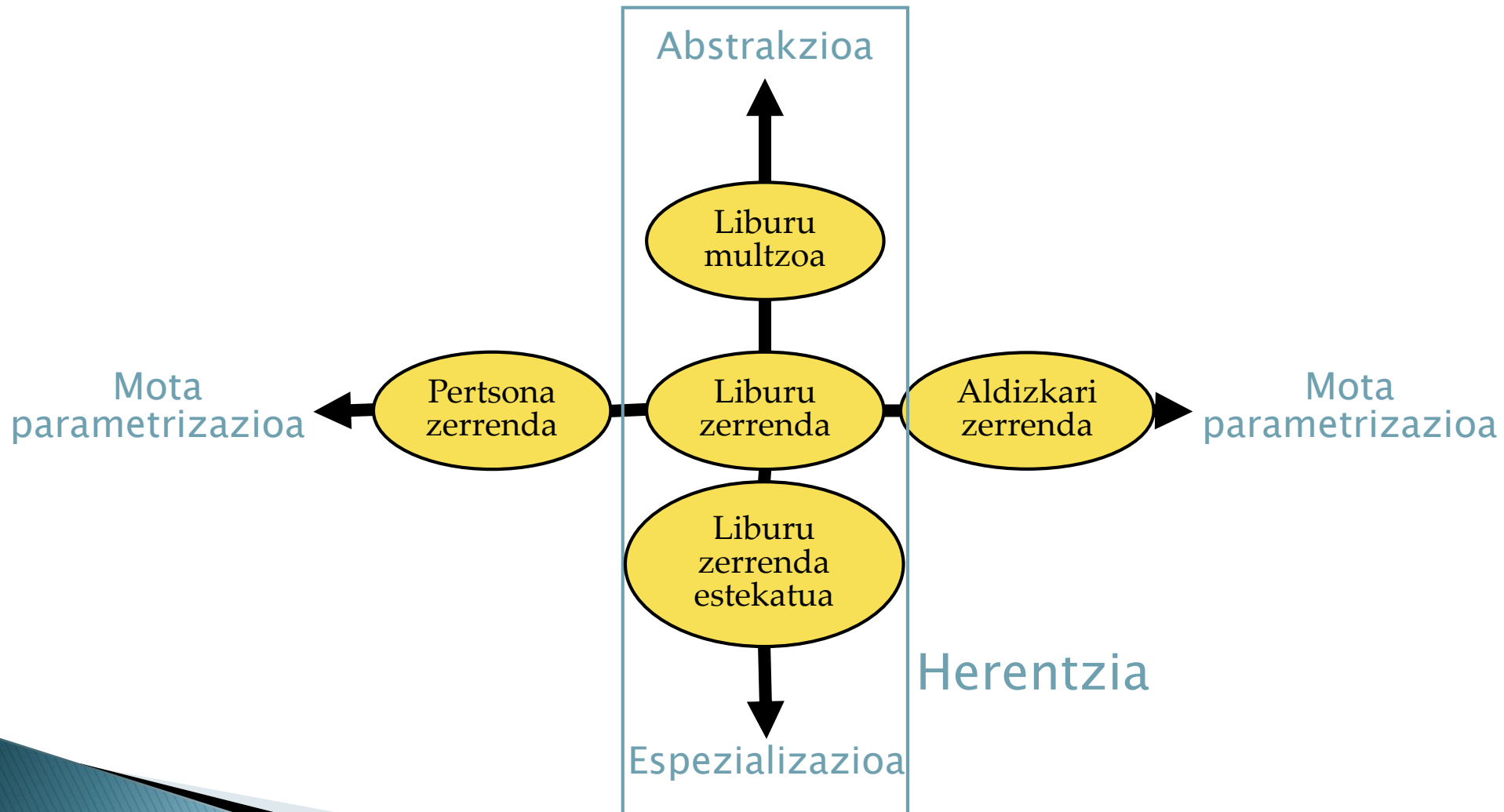
# DEFINIZIOA

- ◉ Generizitatea klase bat bere atributuen datu mota adierazi gabe definitzea posible egiten du. Horrela, kodea ez da berridatzi behar erabilera ezberdinetara moldatzeko.
- ◉ Oso antzerakoak diren klase multzoak daudenean, guztienak diren ezaugarriak batzen dituen klase berezia (**generikoa**) definitu daiteke.
- ◉ **Zer orokortu daiteke?**
  - Datu egiturak (pilak, zuhaitzak, ...)
  - Algoritmoak (ordenazioa, bilaketa, swap, batura,...)

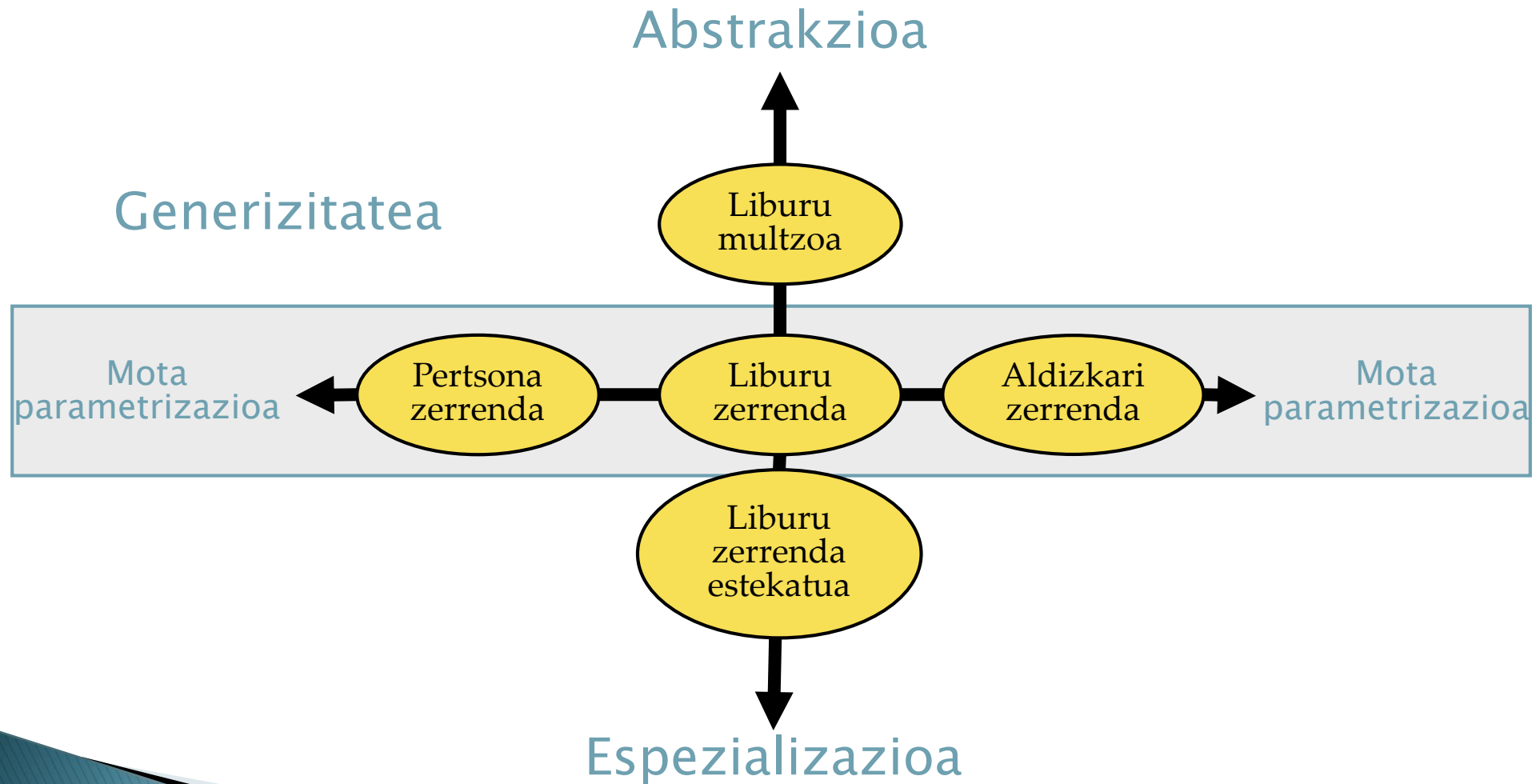
# OROKORTZEAREN DIMENTSIOAK



# OROKORTZEAREN DIMENTSIOAK



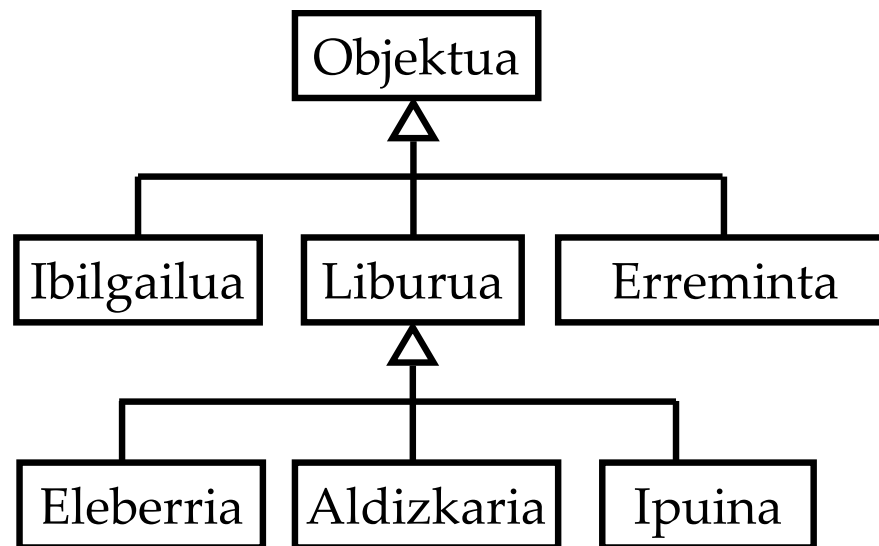
# OROKORTZEAREN DIMENTSIOAK





# DIMENTSIO BERTIKALA

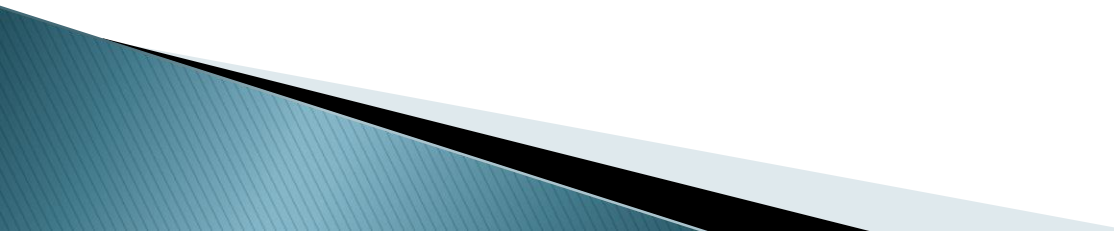
- ▶ Herentziaren mekanismoak erabiltzen ditu.
- ▶ Jokaera berdina duten klaseak antolatu daitezke, arbaso bera duen hierarkia batekin.



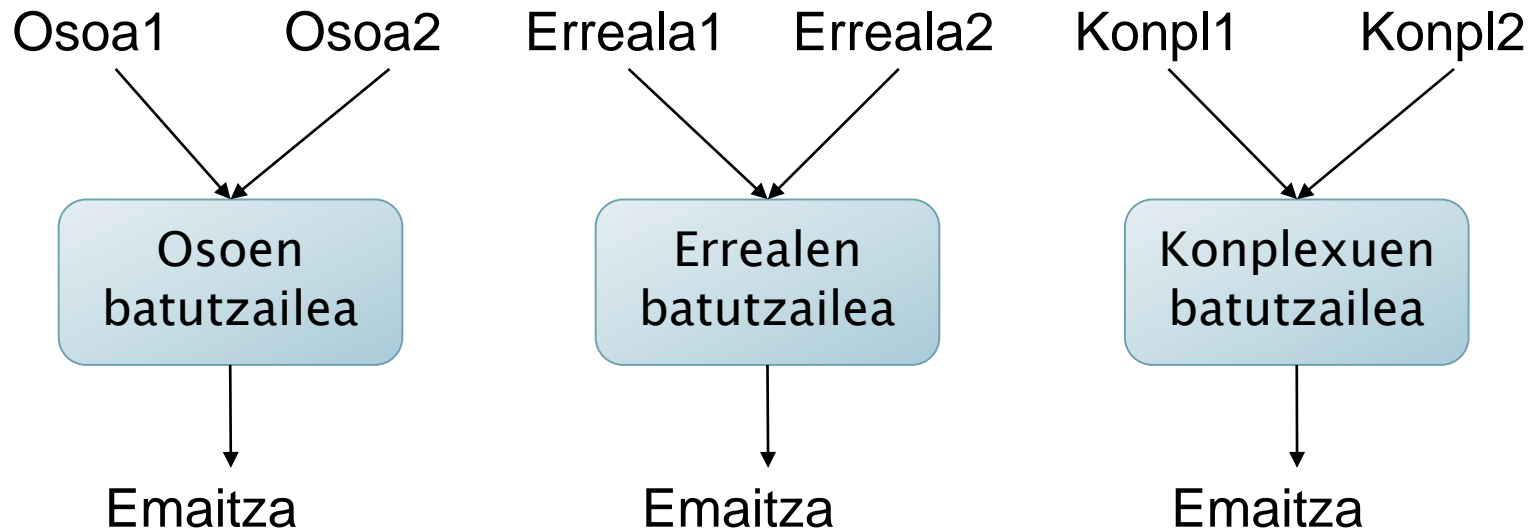
# DIMENTSIO BERTIKALA

- ◉ Hierarkian definitutako klaseen instantziak dituen objektu multzo bat definitu daiteke.
- ◉ Polimorfismo eta lotura dinamikoaren bitartez, Objektu klasearen edo edozein azpiklasearen instantziak izan lituzke.
- ◉ *BAINA...*
  - Nola sortzen da definizioz osagai guztiak mota berekoak dituen zerrenda bat (adibidez, Liburuak)?
  - Zelan definitu zerrenda hori gordetzen dituen osagaien motak aldatu ahal izateko bere kodea aldatu behar gabe?

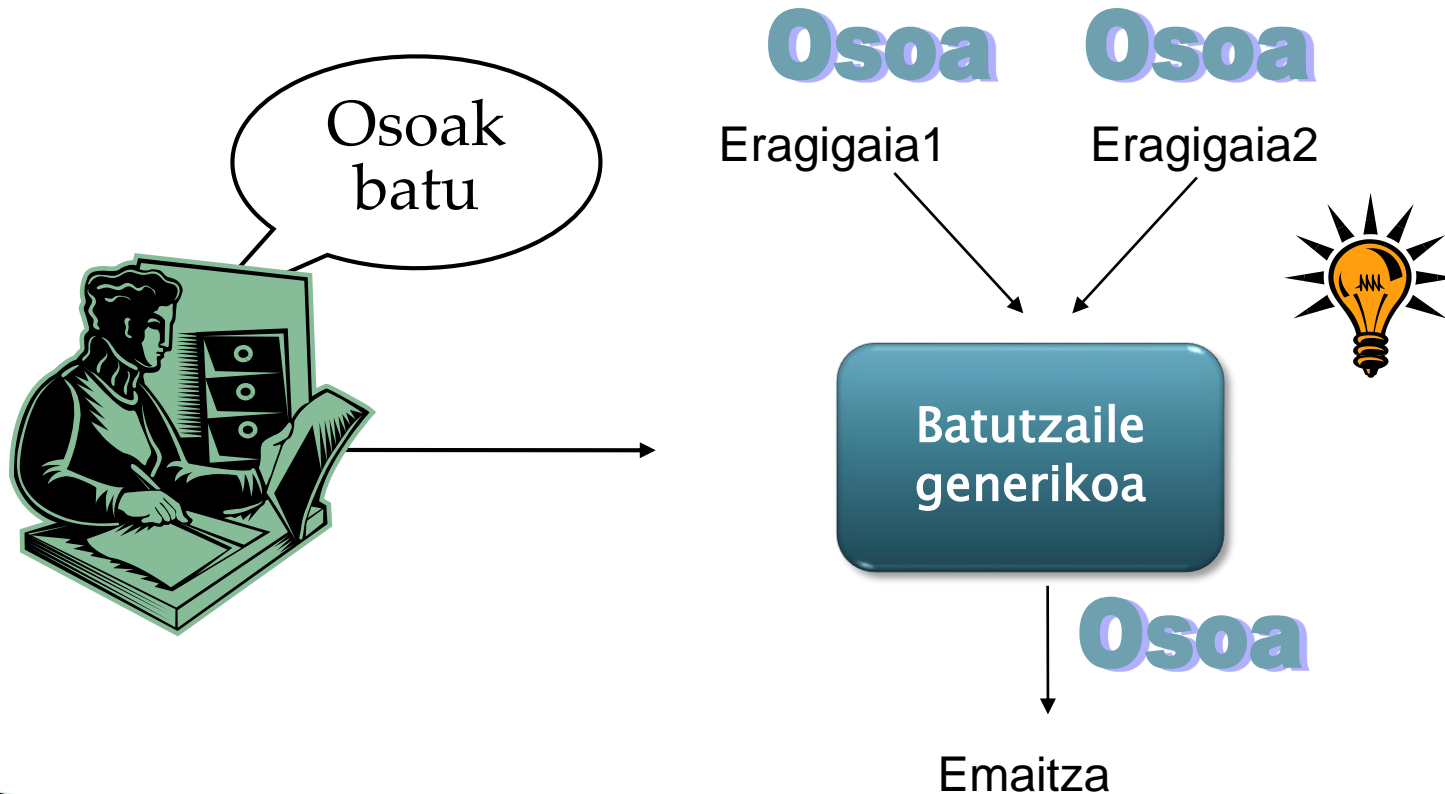
# DIMENTSIO HORIZONTALA

- ▶ Generizitateak motak parametrizatzeko aukera ematen du, tipo ezberdinetarako instantziatu ditzaketen klase eskemak posible eginez.
  - ▶ Oso antzerakoak diren klase multzoak daudenean, guztien ezaugarriak biltzen dituen klase berezi bat (generikoa) definitu daiteke.
- 

# DIMENTSIO HORIZONTALA



# DIMENTSIO HORIZONTALA



# DIMENTSIO HORIZONTALA

- ◉ Guztienak diren portaerak identifikatu
  - Zehaztasunak baztertu eta portaera komuna aprobetxatu
  - Moduluen familia / azpifamiliatan oinarritutako soluzioak inplementzaioaren menpe (adierazpena)

# DIMENTSIO HORIZONTALA

- ◉ Unitate Generikoak (txantiloiak) moduluak sortzeko (objektu edo/eta programak)
- ◉ Bete behar diren “hutsuneak” dituztela esan daiteke
  - Parametro generiko formalak
- ◉ Unitate generikoa instantziazuz zuzenean erabili daiteken modulu bat lortzen da
  - Parametro generiko errealak

# DIMENTSIO HORIZONTALA

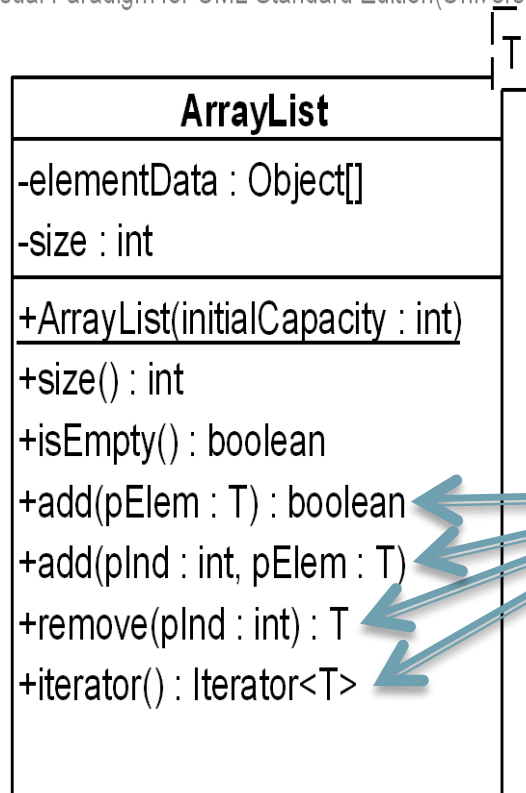
- ◉ Oso antzerakoak diren moduluak sortzeko beharra
  - Osoen zerrendak
  - Errealen zerrendak
  - Ikasle zerrendak
  - ...
  - Osoen pilak
  - Errealen pilak
  - Azterketa pilak
  - ...



# ArrayList

- ▶ JAVA klase generiko baten UML adierazpena

Visual Paradigm for UML Standard Edition(University)



Parametro generiko formala

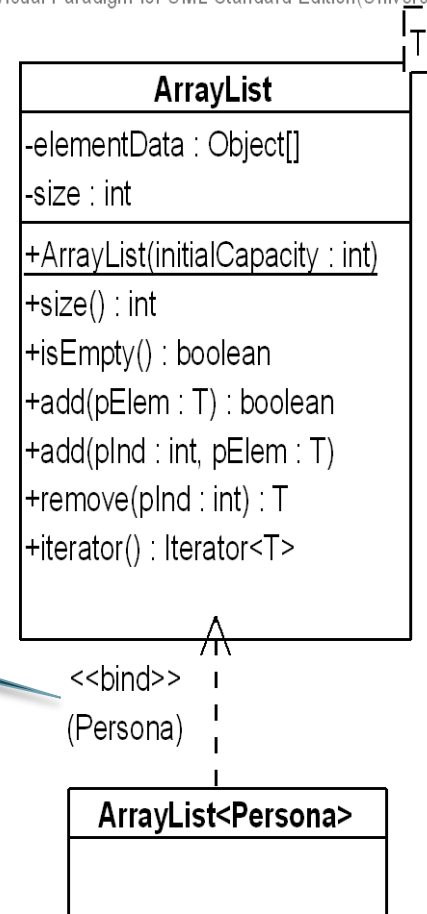
Parametro generiko formala beste edozein mota edo klase bat bezala erabili daiteke

# Pila Generikoa: dimentsio horizontala

- Instantziatze prozesuaren UML adierazpena

Visual Paradigm for UML Standard Edition(University)

ArrayList<Pertsona>  
Parametro generiko  
errealia zehazten da



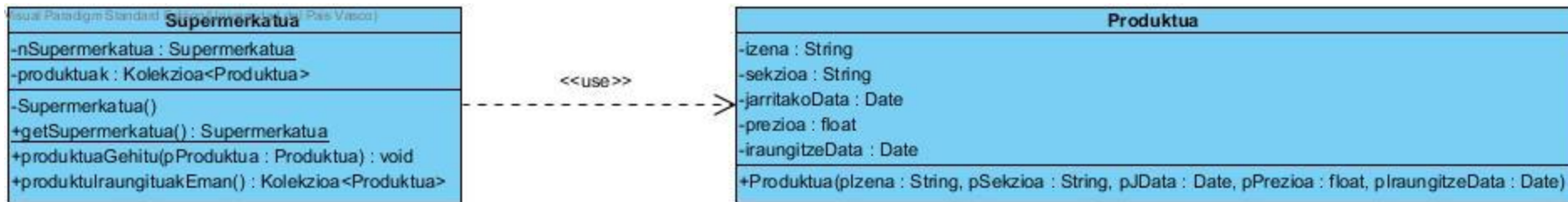
# DIMENTSIO HORIZONTALA

- ▶ Honek berrerabilpen maila bat baimentzen du
- ▶ Java 8n hau aurrerago eramateko aukera dugu

# Software berrerrabilpena Java 8-n

Lambda espresioak eta agregazio operazioak

# Motibazioa

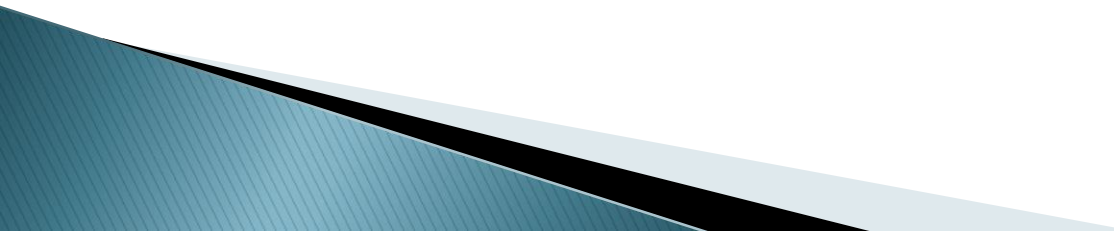


*produktulraungituakEman()* funtzioa inplementatzeko eskatu digute.

# Soluzio posible bat

```
public List<Produktua> produktulraungituakEman () {  
    List<Produktua> iraungituak = new ArrayList<>();  
    LocalDate gaurkoData = LocalDate.now();  
    for (Produktua produktua : produktuak) {  
        if (produktua.getIraungitzeData().isBefore(gaurkoData)) {  
            iraungituak.add(produktua);  
        }  
    }  
    return iraungituak;  
}
```

# Eta orain eskatzen badigute...

- ▶ Bigarren sekzioko produktuen zerrenda?
  - ▶ 2016/02/18 baino beranduago jarri zirenak?
  - ▶ 12 € baino garestiagoak direnak?
  - ▶ ...
- 
- ▶ 10 aldiz antzeko kodea idatzi??
- 

# Behaviour parameterization (I)

- ▶ Baldintza baten arabera filtratzen ari gara
- ▶ Zergatik ez aldatzen dena parametro bezala pasa?

```
public List<Produktua> filtratu (Predicate<Produktua> pPred) {  
    List<Produktua> aukeratuak= new ArrayList<>();
```

```
    for (Produktua produktua : produktua) {  
        if (pPred.test(produktua)) {  
            aukeratuak.add(produktua);  
        }  
    }  
    return aukeratuak;
```

```
}
```



# Behaviour parameterization (II)

```
public List<Produktua> produktulraungituakEman() {  
    return filtratu(new Predicate<>() {  
        public boolean test (Produktua pProd) {  
            return pProd.getIraungitzeData()  
                .isBefore(IsoChronology.INSTANCE.dateNow());  
        }  
    }  
);  
}
```

Klase (anonimo) bat behar dugu baldintza  
adierazteko

# Behaviour parameterization (III)

## ▶ Jada ikusi ditugu

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        ui.dazzle(e.getModifiers());  
    }  
});
```

## ▶ Arazoak:

- Sintaxi konplexua baldintza sinple bat gehitzeko
- Izen eta *this*-ekin konfusioak

# Interfaze funtzionalak

- ▶ Metodo abstraktu bakarra duten interfazeak (ez dira *Object*-etik heredatutako metodoak kontutan hartzen)
- ▶ Funtzioak edo baldintzak irudikatzen dituzte
- ▶ Adibidez

```
@FunctionalInterface
public interface Predicate<T>{
    boolean test(T t);
    //Defektuzko inplementazioa duten beste metodo
    batzuk
}
```

- ▶ `@FunctionalInterface` erabiliz adierazi daitezke
- ▶ `java.util.Function` - en adibide gehiago

# Lambda espresioak

- ▶ Nola erabili interfaze funtzionalak?
- ▶ Orain arte

```
public class IrakasleaDa
    implements Predicate<Pertsona>{
    boolean test(Pertsona pPertsona) {
        return pPertsona.getLana()
            .equals("Irakaslea");
    }
}
```

- ▶ Luzea eta neketsua metodo baten kodea bakarrik idazteko

# Lambda espresioak

- ▶ Interfaze funtzionalak inplementatzen dituzte klase oso bat inplementatu gabe

- ▶ Lambda espresioak

```
p -> p.getLana().equals("Irakaslea")  
p -> p.getIraungitzeData().  
    .isBefore(IsoChronology.INSTANCE.dateNow())
```

- ▶ Sintaxia

- *(parametroak) -> gorputza*
- Parametroak interfaze funtzionalaren metodo abstraktuaren parametro formalen zerrenda dira
  - (Pertsona p)
  - Parametro bakarra badago, parentesiak kendu daitezke
  - Parametro formalaren mota jartzea ez da derrigorrezkoa
- Gorputza instrukzio bloke bat edo espresio bat izan daiteke –  
**Blokeak giltza artean beti**

# Lambda espresioak erabiliz

```
public List<Produktua> produktulraungituakEman() () {  
    return filter(p -> p.getIraungitzeData()  
                    .isBefore(LocalDate.now()));  
}
```

Askoz errazagoa

# Agregazio operazioak (I)

- ▶ Algoritmo arruntenak inplementatzen dituzte
  - Filtratu
  - Map
  - ForEach
  - Batura
  - ...
- ▶ Lambda espresioekin erabiltzen dira
- ▶ Operazioak era konkurrentean egiteko aukera ematen dute

# Agregazio operazioak (II)

## ► Nola?

- Pipeline-ak – operazioak kateatzeko datu fluxu sekuentziak
- Datu fluxuak
  - `stream()` – Sekuentziala
  - `parallelStream()` – Konkurrentea
- Barruan iteratzen dute – ez dute *next* metodoa erakusten
- Erdibideko operazioak
  - Collector-ak: `groupBy`, `filter`, `collect`,...
- Amaierakoak
  - Balio bat itzultzen dute
    - Batura, batazbestekoa...



# Agregazio operazioak: Adibideak (I)

- ▶ Kolekzio bateko elementu guztiak erakutsi

- Sekuentziala

```
list.stream().forEach(p -> System.out.println(p));
```

- Konkurrentea

```
list.parallelStream()  
    .forEach(p -> System.out.println(p));
```

# Agregazio operazioak: Adibideak (II)

```
list.stream()  
    .filter(p -> p.getGender() == Person.Sex.MALE)  
    .mapToInt(Person::getAge)  
    .average()  
    .getAsDouble();
```

# Interfazeak

- ▶ Java 8-n defektuzko implementazio bat gehitu daiteke interfaze baten metodoentzat

```
public interface DoIt {  
    void doSomething(int i, double x);  
    default boolean didItWork(int i, double x, String S)  
    {  
        // Implementazioa  
    }  
}
```

- ▶ Interfazea implementatzen duten klaseek ez dituzte defektuzko implementazioa duten metodoak berriro implementatu behar
- ▶ Interfazeetan metodo estatikoak definitu daitezke (implementazioa izan dezakete)

# Method reference

- ▶ Klase batek interfaze funtzional baten sinadura duen metodo bat badu, parametro bezala pasa daiteke

```
produktuak (comparing (Produktua::getPrezzo) ) ;
```

# Ariketak

1. Pertsonak (nan, izena eta adina) dituen kolekzio bat hartuta, adin nagusikoak direnak inprimatzen dituen programa idatzi
2. Pertsonak adinaren arabera ordenatzen dituen programa idatzi
3. Zenbaki osoen kolekzio bat izanda, 20 baino haundiagoak filtratzen dituen programa idatzi. Eta 30, 40 edo 50 bada?