Java8

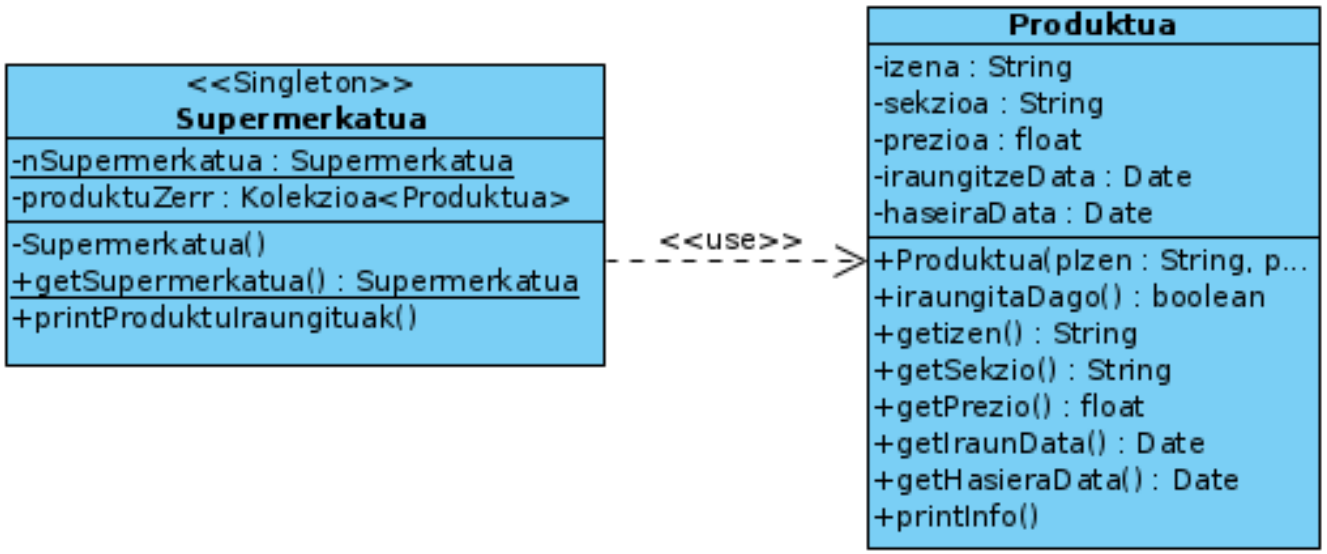
SOFTWARE INGENIARITZA

EDUKIAK

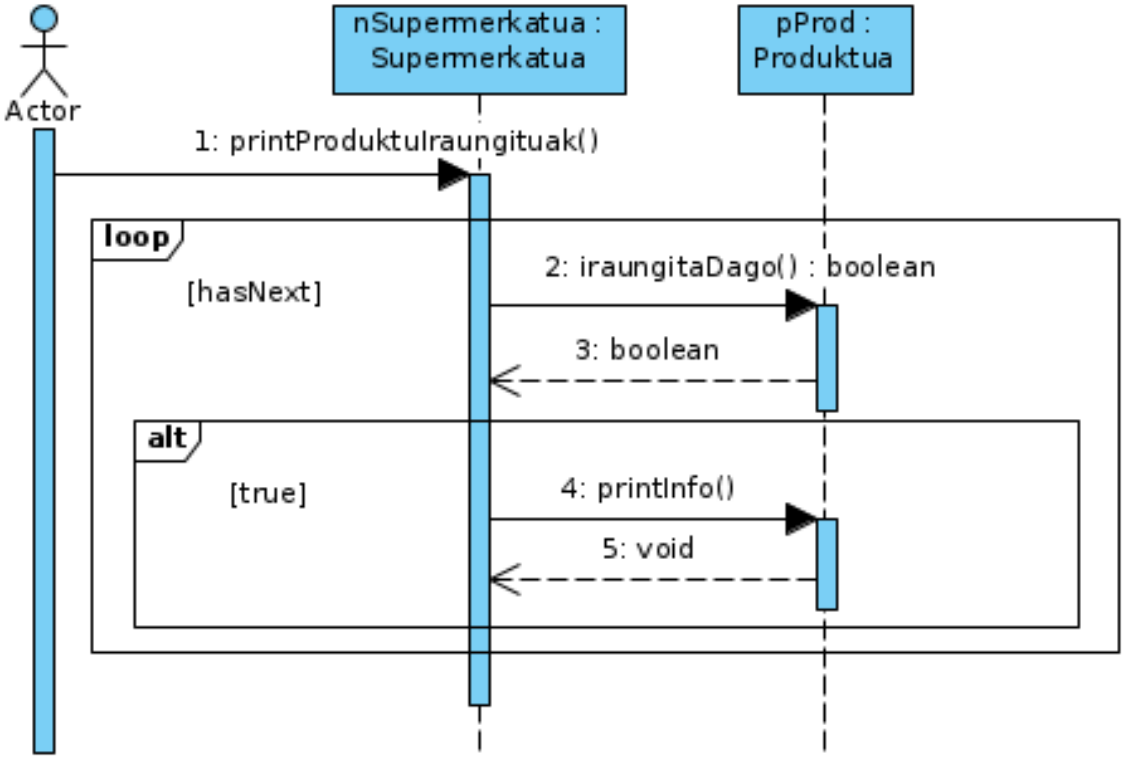
- Sarrera
- Behaviour parametrization
- Interfaze funtzionalak
- Lambda espresioak
- Stream eta agregazio operazioak
- Interfazeak

Sarrera

Supermerkatu klasean `printProduktuIraungituak` kodetzeko eskatu digute. Azken horrek iraungitako produktuak bilatuko ditu eta euren infomazioa pantailaratuko du.



Sekuentzia diagrama



Soluzio posible bat

```
public void printProduktuIraungituak() {
    for (Produktua produktua : produktuZerr) {
        if (produktua.iraungitaDago()){
            produktua.printInfo();
        }
    }
}
```

Hurrengoak eskatuz gero...

Iraungitakoen print:

```
public void printProduktuIraungituak() {
    for (Produktua produktua : produktuZerr) {
       if (produktua.iraungitaDago())
              produktua.printInfo();
     }
}
```

2. sekziokoen print:

```
public void printProduktuSekzio2(){
   for (Produktua produktua : produktuZerr){
      if (produktua.getSekzioa().equals("2"))
            produktua.printInfo();
   }
}
```

12 euro baino garestiagoen print:

```
public void printProduktuKostu() {
 for(Produktua produktua : produktuZerr) {
      if (produktua.getPrezio() > 12)
            produktua.printInfo();
   }
}
```

Hurrengoak eskatuz gero...

Iraungitakoen print:

```
public void printProduktuIraungituak() {
    for (Produktua produktua : produktuZerr) {
       if (produktua.iraungitaDago())
            produktua.printInfo();
    }
}
```

Aldaketa lerro bakarrean, baina hiru metodo!!!

2. sekziokoen print:

```
public void printProduktuSekzio2(){
   for (Produktua produktua : produktuZerr){
      if (produktua.getSekzioa().equals("2"))
          produktua.printInfo();
   }
}
```

12 euro baino garestiagoen print:

```
public void printProduktuKostu() {
 for(Produktua produktua : produktuZerr) {
     if (produktua.getPrezio() > 12)
         produktua.printInfo();
   }
}
```

Hurrengoak eskatuz gero...

Iraungitakoen print:

```
public void printProduktuIraungituak() {
    for (Produktua produktua : produktuZerr) {
        if (produktua.iraungitaDago())
            produktua.printInfo();
    }
}
```

Betekizunen aldaketen aurrean, nola berrerabili antzeko kodea?

```
public void printProduktuSekzio2(){
    for (Produktua produktua : produktuZerr){
        if (produktua.getSekzioa().equals("2"))
            produktua.printInfo();
    }
}
```

```
public void printProduktuKostu() {
    for(Produktua produktua : produktuZerr) {
        if (produktua.getPrezio() > 12)
            produktua.printInfo();
    }
}
```

Behaviour parametrization

```
public interface Filtratu {
        boolean test(Produktua pProd);
}
```

Interfazea

```
public class Iraungitakoak
                    implements Filtratu{
   @Override
   public boolean test(Produktua pProd) {
          return pProd.iraungitaDago();
     }
}
```

Interfazearen 1. inplementazioa

```
public class Sekziokoak
                    implements Filtratu{
   @Override
   public boolean test(Produktua pProd) {
          return pProd.getSekzio().equals("2");
     }
}
```

Interfazearen 3. inplementazioa

```
public class Kostukoak
                    implements Filtratu{
   @Override
   public boolean test(Produktua pProd) {
          return pProd.getPrezio()>12;
     }
}
```

Interfazearen 2. inplementazioa

Aldatzen dena parametro legez. Interfazeak!!!

Behaviour parametrization

```
public void filtratuProd(Filtratu pFiltro) {
     for (Produktua produktua : produktuZerr) {
          if (pFiltro.test(produktua))
               produktua.printInfo();
     }
}
```

Supermerkatua Klasea

```
public interface Filtratu {
        boolean test(Produktua pProd);
}
```

Interfazea

```
public class Iraungitakoak implements Filtratu
{...}
public class Sekziokoak implements Filtratu
{...}
public class Kostukoak implements Filtratu
{...}
```

Interfazearen inplementazioak

```
superM.filtratuProd(new Iraungitakoak());
superM.filtratuProd(new Sekziokoak());
superM.filtratuProd(new Kostukoak());
```

MAIN

Aldatzen dena parametro legez. Interfazeak!!!

Behaviour parametrization

```
public void filtratuProd(Filtratu pFiltro) {
     for (Produktua produktua : produktuZerr) {
          if (pFiltro.test(produktua))
               produktua.printInfo();
     }
}
```

Supermerkatua Klasea

```
public interface Filtratu {
        boolean test(Produktua pProd);
}
```

Interfazea

```
public class Iraungitakoak implements Filtratu
{...}
public class Sekziokoak implements Filtratu
{...}
public class Kostukoak implements Filtratu
{...}
```

Interfazearen inplementazioak

```
superM.filtratuProd(new Iraungitakoak());
superM.filtratuProd(new Filtratu(){
                        @Override
                            public boolean test(Produktua pProd) {
                             return pProd.getSekzio().equals("2");
                        }
                    }
                );
```

Klase anonimoa

MAIN

Aldatzen dena parametro legez. Interfazeak!!!

Interfaze funtzionalak

- **Java8**-tik aurrera, **aurredefinitutako** interfazeak dira:
  - **Metodo abstraktu bakarra**
- Funtzioak/baldintzak irudikatzen dituzte: **portaerak**
- Definitzerakoan, `@FunctionalInterface` jarri

Predicate

```
@FunctionalInterface
public interface Predicate <T>{
    boolean test (T t) ;
}
```

Consumer

```
@FunctionalInterface
public interface Consumer <T>{
    void accept (T t) ;
}
```

Supplier

```
@FunctionalInterface
public interface Supplier <T>{
    T get () ;
}
```

Function

```
@FunctionalInterface
public interface Function <T,R>{
    R apply (T t) ;
}
```

Interfaze funtzionalak

```
public void filtratuProd (Predicate<Produktua> pPredicate) {
  for (Produktua produktua : produktuZerr) {
    if (pPredicate.test(produktua))
          produktua.printInfo();
    }
  }
}
```

Supermerkatua

Inplementazioak

```
public class Iraungitakoak implements Predicate<Produktua>{...}

public class Sekziokoak implements Predicate<Produktua>{...}

public class Kostukoak implements Predicate<Produktua>{...}
```

```
public interface Predicate <T>{
    boolean test (T t) ;
}
```

```
superM.filtratuProd(new Iraungitakoak());
superM.filtratuProd(new Sekziokoak());
superM.filtratuProd(new Kostukoak());
```

MAIN

Baina, oraindik inplementazioa egin behar!

Klase berri bat edo klase anonimoa...

Lambda espresioak

- Nola erabili interfaze funtzionalak?
- Orain arte

```
public class Sekziokoak implements Predicate<Produktua>{
    boolean test(Produktua pProduktua) {
         return pProduktua.getSekzio().equals("2");
    }
}
```

- Luzea eta neketsua

Lambda espresioak

```
public class Sekziokoak implements Predicate<Produktua>{
    boolean test(Produktua pProduktua) {
        return pProduktua.getSekzio().equals("2");
    }
}
```

Askoz konpaktuagoa!!!

```
p -> p.getSekzio().equals("2")
```

Lambda espresioak

```
public class Sekziokoak implements Predicate<Produktua>{
    boolean test(Produktua pProduktua) {
        return pProduktua.getSekzio().equals("2");
    }
}
```

Sarrera parametroa

```
p -> p.getSekzio().equals("2")
```

Lambda espresioak

```
public class Sekziokoak implements Predicate<Produktua>{
    boolean test(Produktua pProduktua) {
        return pProduktua.getSekzio().equals("2");
    }
}
```

inplementazioa

```
p -> p.getSekzio().equals("2")
```

Lambda espresioak

- Interfaze funtzionalak inplementatu, klaserik sortu barik

```
p -> p.getSekzio().equals("2")
p -> p.iraungitaDago()
p -> p.getPrezio() > 12
```

- Parametroak egitekoekin erlazionatzen dituzte

Lambda espresioak

- Sintaxia

```
(parametroak) -> {gorputza}
```

  - *Parametroak:* interaze funtzionalaren metodo abstraktuaren parametro zerrenda

Parametro bakarrarekin, ez da zertan parentesirik jarri behar

```
p          -> p.getSekzio().equals("2")
( p , pr)  -> {p.getPrezio() > pr}
```

  - *Gorputza:* instrukzio blokea edo espresioa – **giltz artean**

```
p          -> p.getSekzio().equals("2")
( p , pr)  -> {p.getPrezio() > pr}
```

Instrukzio bakarrarekin, ez da zertan giltzik jarri behar

Lambda espresioak

```
public void filtratuProd (Predicate<Produktua>  pPredicate) {
     for (Produktua produktua : produktuZerr) {
          if (pPredicate.test(produktua))
               produktua.printInfo();
     }
}
```

```
public interface Predicate <T>{
     boolean test (T t) ;
}
```

Supermerkatua

```
superM.filtratuProd( p -> p.iraungitaDago() );
superM.filtratuProd( p -> p.getSekzioa().equals("2"));
superM.filtratuProd( p -> p.getPrezio() > 12 );
```

MAIN

Inplementazioa (portaera) parametro legez pasatu, lambda espresio bidez

Metodo erreferentziak

- Klase batek interfaze funtzional baten sinadura daukan metodoa badu, metodoaren erreferentzia parametro bezala pasa daiteke.
- Sintaxia:

```
Klasea::metodoa
```

- Adibidea:

```
produktuak(comparing(p->p.getPrezioa()));
produktuak(comparing(Produktua::getPrezioa));
```

Metodo erreferentziak

```
Consumer<String> cons = p -> System.out.println(p);

Consumer<String> cons =      System.out::println;
```

```
return produktuZerrenda.stream()
    .collect(partitioningBy( p -> p.iraungitaDago(),
                             averagingDouble(p -> p.getPrezio()));
}

return produktuZerrenda.stream()
    .collect(partitioningBy( Produktua::iraungitaDago,
                             averagingDouble(Produktua::getPrezio));
}
```

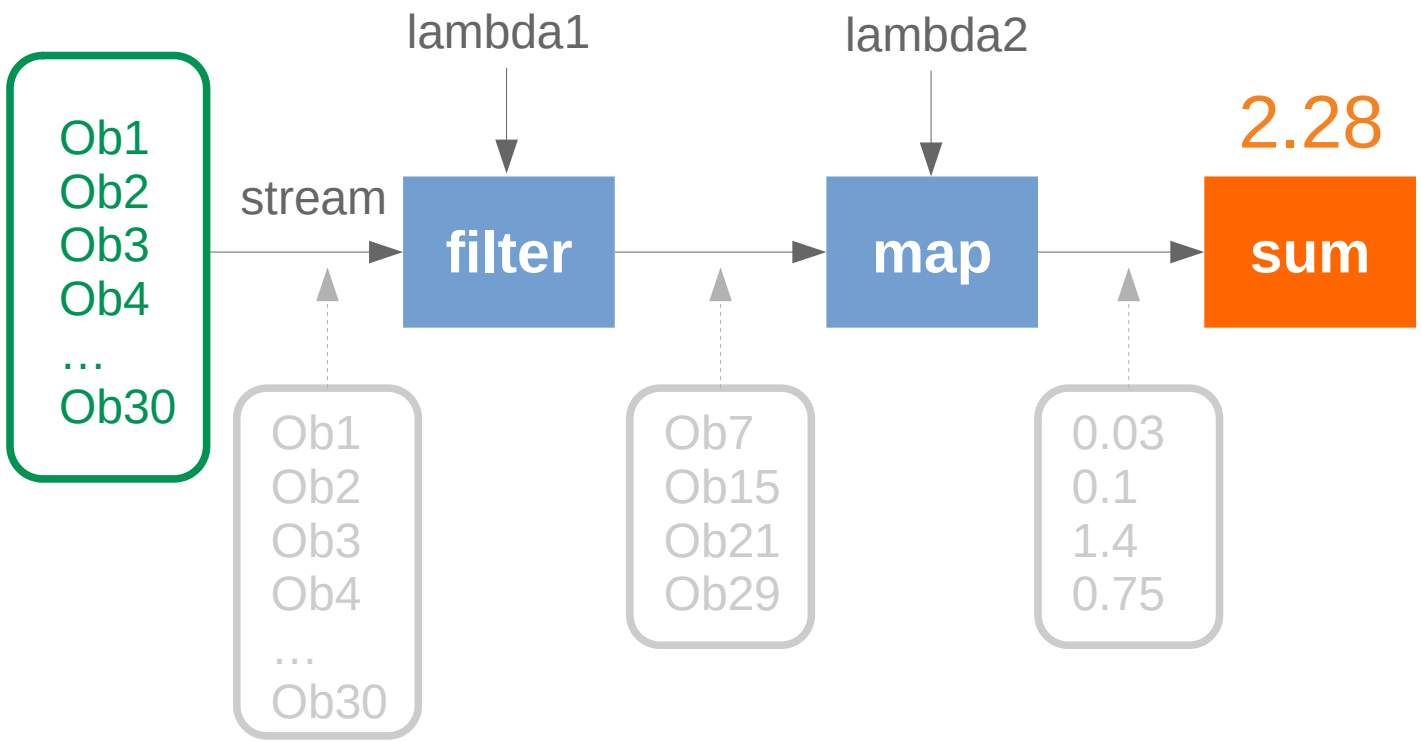
Stream eta agregazio operazioak

- Java-ren 8. bertsioko nobedadeak:
  - Algoritmo arruntenen inplementazioa
    - Filtraketa
    - Map
    - ForEach
    - Batura
    - ….
  - Lambda espresioen erabilpena
  - Barne iterazioak *Stream*-en bidez

Stream eta agregazio operazioak

- Nola dabiltza berritasun horiek?
  - *Pipeline:* operazio kateaketa (datu fluxu sekuentzia)
    - Datu fluxuak
      - `stream()` : Sekuentziala
      - `parallelStream()` : Konkurrentea
  - *Barne iterazioak*
  - *Operazioak:*
      - Bitartekoak: `map`, `sorted`, `filter` ...
      - Amaierakoak: `collect`, `sum`, `forEach` ...
  -

Stream eta agregazio operazioak



Stream eta agregazio operazioak

```
public void printProduktuIraungituak() {

for (Produktua produktua : produktuak) {
  if (produktua.iraungitaDago())
produktua.printInfo();
}

}
```

JAVA7

Stream eta agregazio operazioak

```
public void printProduktuIraungituak() {

    for (Produktua produktua : produktuak) {
      if (produktua.iraungitaDago())
            produktua.printInfo();
    }

}
```

JAVA7

```
Public List<Produktua> printProduktuIraungituak(){
  return produktuak.stream()                    //barne iterazioa
              .filter(iraungitaDago())    //filtroa
              .forEach(Produktua::printInfo);//banan-bana
                                              //pantailaratu
}
```

JAVA8

Stream eta agregazio operazioak

- Barne iterazioa:
  - Sekuentziala

```
produktuZerrenda.stream()
          .filter( p -> p.iraungitaDago())
          .forEach(Produktua::printInfo);
```

  - Paraleloa

```
produktuZerrenda.parallelStream()
          .filter( p -> p.iraungitaDago())
          .forEach(Produktua::printInfo);
```

Stream eta agregazio operazioak

- **`stream` *VS* `parallelStream`**

`parallelStream`-ek datuen fluxua prozesadoreak beste zatitan banatzen du. Elementuen prozesaketaren ordena aldatu egin daiteke.

Stream eta agregazio operazioak

- **Bitarteko operazioak**: fluxu berria sortu

OP	Argumentua	Buelta	Helburua
`filter`	`Predicate<T>`	`Stream<T>`	Predikatua betetzen duten elementuen fluxua bueltatu.
`map`	`Function<T,R>`	`Stream<R>`	Fluxuko elementu bakoitzari funtzio bat aplikatu, eta emaitza fluxu berri batean bueltatu. Tipo primitoboentzat aldaerak daude (`mapToInt` edo `mapToDouble`)
`sorted`	`Comparator<T>`	`Stream<T>`	Fluxu bateko elementuak baldintza batzuen arabera ordenatu eta emaitza fluxu berri batean bueltatu.
`distinct`		`Stream<T>`	Fluxu berria bueltatu, errepikatu gabeko elementuez osatutakoa

Stream eta agregazio operazioak

- Bitarteko operazioak:

```
Public double getIraungituenPrezioTotala(){
  return produktuZerrenda.stream()
            .filter(p->p.iraungitaDago())     //filtroa
            .mapToDouble(p->p.getPrezio())   //mapaketa
            .sum();                          //batuketa
}
```

```
Public List<Produktua> getZerrendaPreziozOrdenatuta(){
  return produktuZerrenda.stream()
          .mapToDouble(p->p.getPrezio())       //mapaketa
          .sorted((i1,i2)-> i2.compareTo(i1)) //konparaketa
          .collect(toList());                  //bilketa
}
```

Stream eta agregazio operazioak

- **Amaierako operazioak:** prozesua ejekutatu

OP	Argumentua	Buelta	Helburua
forEach	`Consumer<T>`	`void`	Fluxuko elementu bakoitza kontsumitu, definitutako lambda aplikatuz.
count		`long`	Fluxuko elementu kopurua bueltatu.
collect	`Collector<T,A,R>`	`R`	Fluxua erreduzitu zerrenda mapa edo balio oso bat sortzeko, definitutako rekolekzio metodoaren arabera.
anyMatch	`Predicate<T>`	`boolean`	Fluxuko elementuetako batek predikatua betetzen badu, `true` bueltatu.
allMatch	`Predicate<T>`	`boolean`	Fluxuko elementu orok predikatua betetzen badute, `true` bueltatu.

Stream eta agregazio operazioak

- **Amaierako operazioak**: zenbakidun fluxuak (`IntStream` edo `DoubleStream`)

OP	Arg.	Buelta	Helburua
sum		`int` edo `double`	Fluxuko elementuen batuketa bueltatu.
average		`OptionalDouble`	Fluxuko elementuen batazbestekoa bueltatu.
summaryStatsitics		`IntSummaryStatistics`, `DoubleSummaryStatistics`	Fluxuko elementuen estatistikak bueltatzen ditu

Stream eta agregazio operazioak

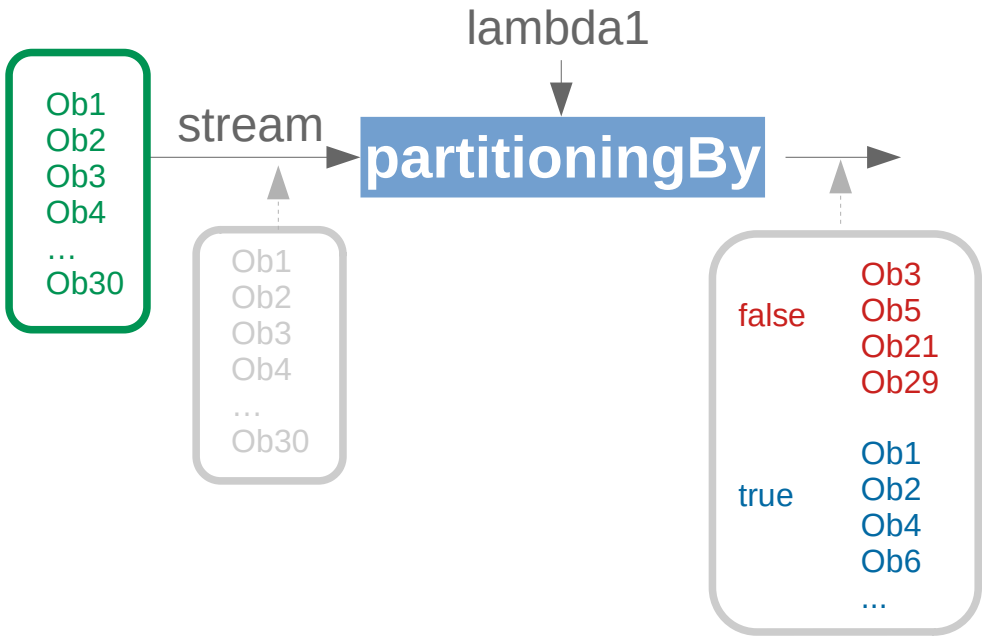
- Bilketa metodoak: modu estatikoan inportatzea komeni. `java.util.stream.Collectors` klasea.

OP	Argumentua	Buelta	Helburua
`toList`		`int`	Fluxu bateko elementuak biltzen dituen kolektorea bueltatu.
`partitioningBy`	`Predicate<T>`	`Map<boolean,D>`	Predikatu baten arabera, elementuak (erredukzioa aplikatuz) biltzen dituen kolektorea bueltatu.
`groupingBy`	`Function<T>`	`Map<K,D>`	Sailkapen baten arabera, elementuak (erredukzioa aplikatuz) biltzen dituen kolektorea bueltatu.

Stream eta agregazio operazioak

- Bilketa metodoak:

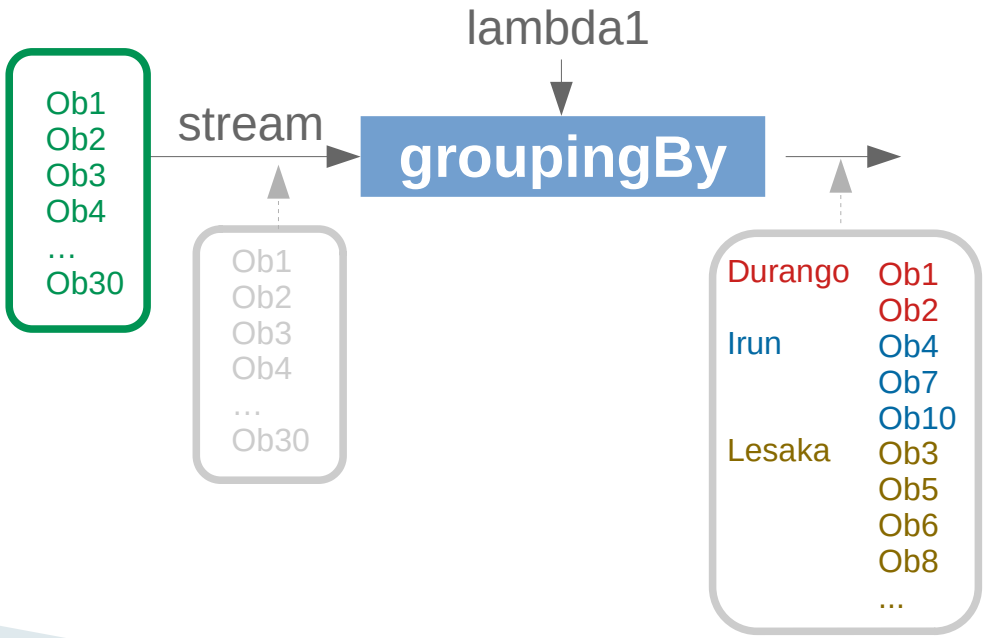
```
Public Map<Boolean,List<Produktu>> getIraungiEziraungiZerr(){
  return produktuZerrenda.stream()
    .collect(partitioningBy(p->p.iraungitaDago());

}
```



Stream eta agregazio operazioak

- Bilketa metodoak:

```
Public Map<String,List<Produktu>> getSekzioZerr(){
  return produktuZerrenda.stream()
    .collect(groupingBy(p->p.getSekzio());
}
```



Stream eta agregazio operazioak

- **`Optional`-ak**:
  - Motibazioa:
    - Zein da sekuentzi huts baten batazbestekoa?
    - Ezein elementuk bilaketa irizpiderik bete ezean, zer bueltatu?
  - **`Optional<T>`** : Balio bat enkaptsulatzeko datu mota, existitzen baldin bada.
    - Metodoak ditu:
      - hutsik dagoen jakiteko: `isPresent`
      - balioa eskatzeko: `get`
      - defektuzko balioa hutsik badago: `orElse`
    - Tipo primitoboentzako inplementazioak (`OptionalDouble`...)

Interfazeak

- Java8-n *defektuzko inplementazio* bat gehitu daiteke
  - `implements` egiten duten klaseek ez dute defektuzko inplementazioekin ezer egin behar

```
public interface DoIt{
    void doSomething(int i, double x);
    default void defektuzkoMetodoa(){
        System.out.println("Defektuzko metodoa naiz!");
    }
}
```

- Interfazeetan *metodo estatikoak* definitu daitezke
  - Ezin dira deitu `implements` egiten duten klaseetatik, interfazearen izenetik baizik