


# Java8

SOFTWARE INGENIARITZA

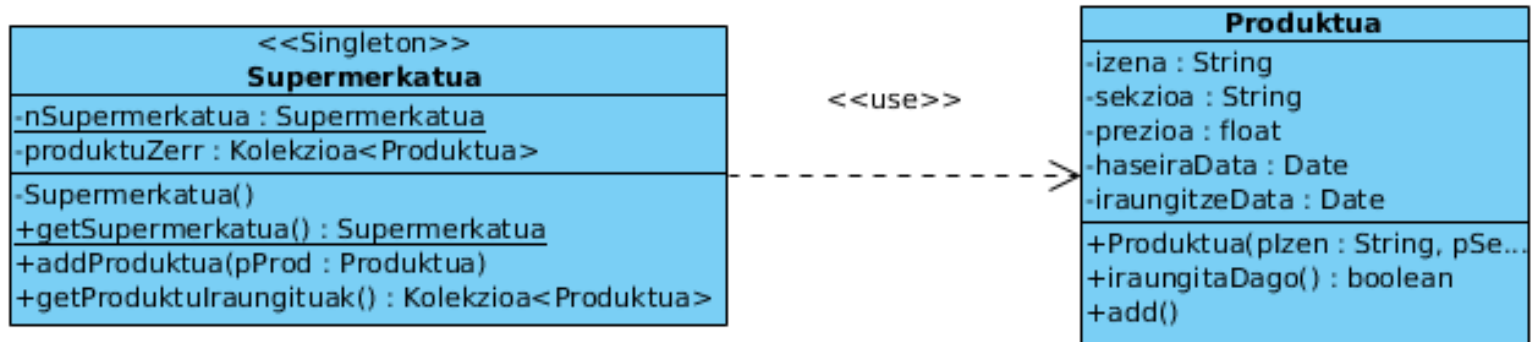


# EDUKIAK

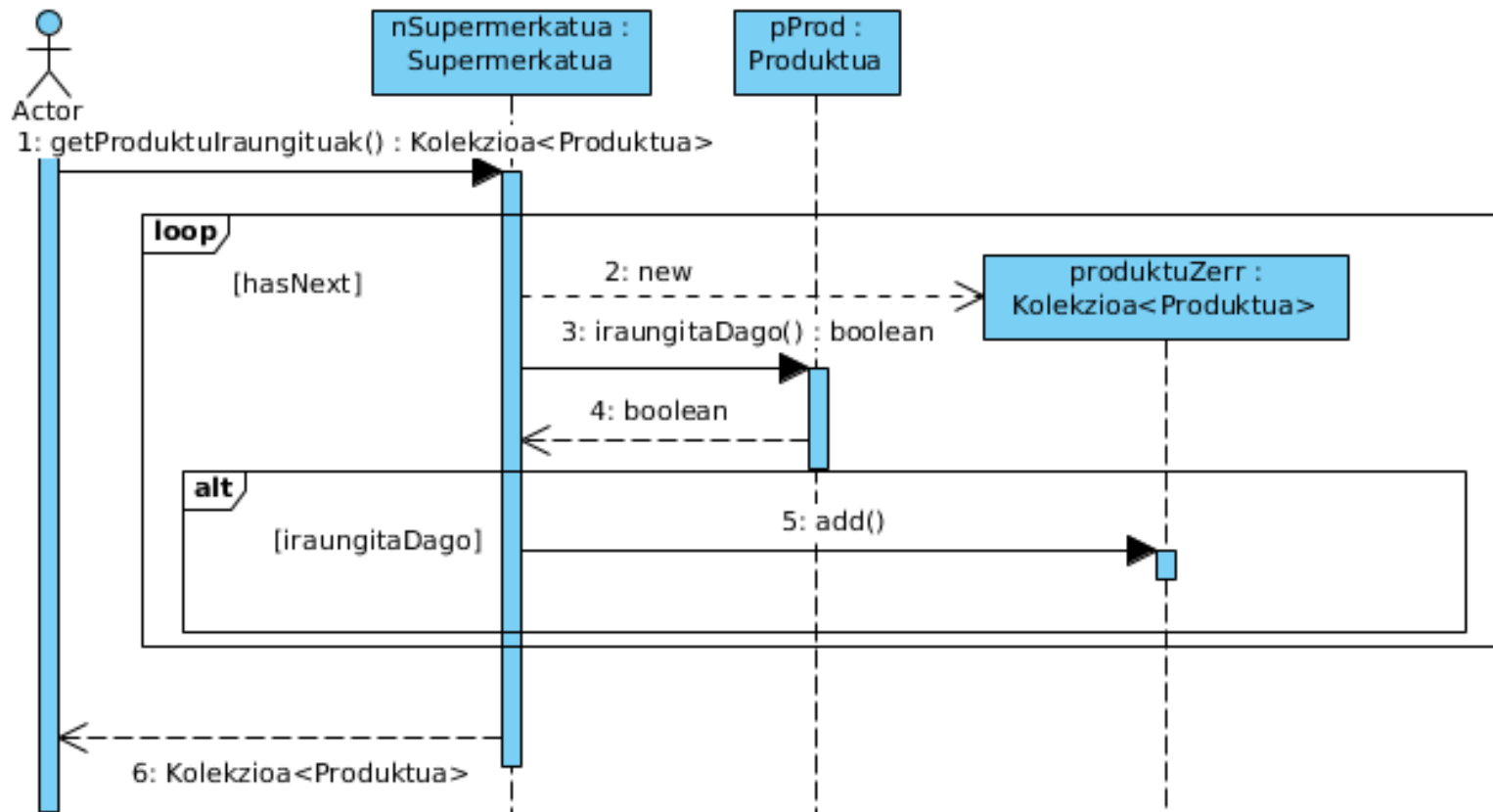
- ▶ Sarrera
  - ▶ Behaviour parametrization
  - ▶ Interfaze funtzionalak
  - ▶ Lambda espresioak
  - ▶ Stream eta agregazio operazioak
  - ▶ Interfazeak
- 

# Sarrera

**Supermerkatu** klasean *produktulraungituakEman()* kodetzeko eskatu digute. Azken horrek iraungitako produktuen zerrenda bueltatzen du.




# Sekuentzia diagrama



# Soluzio posible bat

```
public List<Produktua> getProduktuIraungituak() {  
    List<Produktua> iraungituak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.iraungitaDago()) {  
            iraungituak.add(produktua);  
        }  
    }  
    return iraungituak;  
}
```



# Hurrengoak eskatuz gero...

## Iraungitakoen zerrenda:

```
public List<Produktua> getProduktuIraungituak() {  
    List<Produktua> iraungituak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.iraungitaDago())  
            iraungituak.add(produktua);  
    }  
    return iraungituak;  
}
```

## 2. sekziokoen zerrenda:

```
public List<Produktua> getProduktuSekzio2(){  
    List<Produktua> sekziokoak =newArrayList<>();  
    for (Produktua produktua : produktuak){  
        if (produktua.getSekzioa().equals("2"))  
            sekziokoak.add(produktua);  
    }  
    return sekziokoak;  
}
```

## 12 euro baino garestiagoen zerrenda:

```
public List<Produktua> getProduktuKostu() {  
    List<Produktua>kostukoak =newArrayList<>();  
    for(Produktua produktua : produktuak) {  
        if (produktua.getPrezio() > 12)  
            kostukoak.add(produktua);  
    }  
    return kostukoak;  
}
```

# Hurrengoak eskatuz gero...

## Iraungitakoen zerrenda:

```
public List<Produktua> getProduktuIraungituak() {  
    List<Produktua> iraungituak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.iraungitaDago())  
            iraungituak.add(produktua);  
    }  
    return iraungituak;  
}
```

Aldaketa lerro  
bakarrean, baina  
hiru metodo!!!

## 2. sekziokoen zerrenda:

```
public List<Produktua> getProduktuSekzio2() {  
    List<Produktua> sekziokoak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.getSekzioa().equals("2"))  
            sekziokoak.add(produktua);  
    }  
    return sekziokoak;  
}
```

## 12 euro baino garestiagoen zerrenda:

```
public List<Produktua> getProduktuKostu() {  
    List<Produktua> kostukoak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.getPrezio() > 12)  
            kostukoak.add(produktua);  
    }  
    return kostukoak;  
}
```

# Hurrengoak eskatuz gero...

## Iraungitakoen zerrenda:

```
public List<Produktua> getProduktuIraungituak() {  
    List<Produktua> iraungituak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if ((produktua.iraungitaDago()))  
            iraungituak.add(produktua);  
    }  
    return iraungituak;  
}
```

**Betekizunen aldaketen aurrean,  
nola berrerabili antzeko kodea?**

```
public List<Produktua> getProduktuSekzio2(){  
    List<Produktua> sekziokoak =newArrayList<>();  
    for (Produktua produktua : produktuak){  
        if ((produktua.getSekzioa().equals("2")))  
            sekziokoak.add(produktua);  
    }  
    return sekziokoak;  
}
```

```
public List<Produktua> getProduktuKostu() {  
    List<Produktua>kostukoak =newArrayList<>();  
    for(Produktua produktua : produktuak) {  
        if ((produktua.getPrezio() > 12))  
            kostukoak.add(produktua);  
    }  
    return kostukoak;  
}
```



# Behaviour parametrization

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

**Interfazea**

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

**Interfazearen 3. implementazioa**

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

**Interfazearen 1. implementazioa**

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

**Interfazearen 2. implementazioa**

**Aldatzen dena parametro legez. Interfazeak!!!**

# Behaviour parametrization

```
public List<Produktua> filtratuProd(Filtratu pFiltro,
{
    List<Produktua> filtratuak= new ArrayList<>();
    for (Produktua produktua : produktuZerr) {
        if (pFiltro.test(produktua))
            filtratuak.add(produktua);
    }
    return filtratuak;
}
```

**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 implementazioak**

```
List<Produktua> aIraungi=superM.filtratuProd(new Iraungitakoak());
List<Produktua> aSekzio =superM.filtratuProd(new Sekziokoak());
List<Produktua> aKostu  =superM.filtratuProd(new Kostukoak());
```

**MAIN**

**Aldatzen dena parametro legez. Interfazeak!!!**

# Behaviour parametrization

```
public List<Produktua> filtratuProd(Filtratu pFiltro,
{
    List<Produktua> filtratuak= new ArrayList<>();
    for (Produktua produktua : produktuZerr) {
        if (pFiltro.test(produktua))
            filtratuak.add(produktua);
    }
    return filtratuak;
}
```

**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 implementazioak**

```
List<Produktua> aIraungi=superM.filtratuProd(new Iraungitakoak());
List<Produktua> aSekzio=superM.filtratuProd(new Filtratu(){
```

**Klase anonimoa**

**MAIN**

@Override

```
public boolean test(Produktua pProd) {
    return pProd.getSekzio().equals("2");
}}
```

**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 List<Produktua> filtratuProd (Predicate<Produktua> pPredicate) {  
    List<Produktua> filtratuak= new ArrayList<>();  
    for (Produktua produktua : produktuZerr) {  
        if (pPredicate.test(produktua))  
            filtratuak.add(produktua);  
    }  
    return filtratuak;  
}
```

**Supermerkatua**

**Implementazioak**

```
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) ;  
}
```

```
List<Produktua> aIraungi = superM.filtratuProd(new Iraungitakoak());  
List<Produktua> aSekzio  = superM.filtratuProd(new Sekziokoak());  
List<Produktua> aKostu   = 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<Pertsona>{  
    boolean test(Produktua pProduktua) {  
        return pProduktua.getSekzio().equals("2");  
    }  
}
```

- ▶ Luzea eta neketsua

# Lambda espresioak

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

**Askoz konpaktuagoa!!!**

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



# Lambda espresioak

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

Sarrera parametroa



**p**

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



# Lambda espresioak

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

implementazioa

```
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

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

- *Gorputza*: instrukzio blokea edo espresioa - **giltza artean**

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

# Lambda espresioak

```
public List<Produktua> filtratuProd (Predicate<Produktua> pPredicate) {  
    List<Produktua> filtratuak= new ArrayList<>();  
    for (Produktua produktua : produktuZerr) {  
        if (pPredicate.test(produktua))  
            filtratuak.add(produktua);  
    }  
    return filtratuak;  
}
```

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

**Supermerkatua**

```
List<Produktua> aIraungi=superM.filtratuProd( (p) -> p.iraungitaDago() );  
List<Produktua> aSekzio =superM.filtratuProd( (p) -> p.getSekzioa().equals("2") );  
List<Produktua> aKostu =superM.filtratuProd( (p) -> p.getPrezio() > 12 );
```

**MAIN**

**Implementazioa (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`

`Objektua::metodoa`

- ▶ Adibidea:



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


# Metodo erreferentziak

```
List<Produktua> aIraungi=superM.filtratuProd( (p) -> p.iraungitaDago() );
```


```
List<Produktua> aIraungi=superM.filtratuProd( Produktua::iraungitaDago );
```

```
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

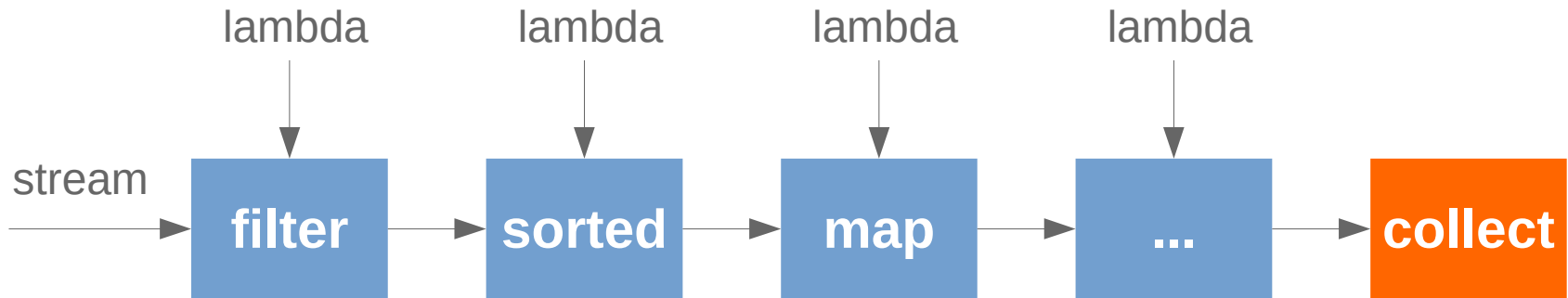
- ▶ Java-ren 8. bertsioiko nobedadeak:
  - Algoritmo arruntaren inplementazioa
    - Filtraketa
    - Map
    - ForEach
    - Batura
    - ....
  - Lambda espresioen erabilpena
  - Barne iterazioak *Stream*-en bidez 

# Stream eta agregazio operazioak

- ▶ Nola dabilta berrietasun 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 List<Produktua> getProduktuIraungituak() {  
    List<Produktua> iraungituak = new ArrayList<>();  
    for (Produktua produktua : produktuak) {  
        if (produktua.iraungitaDago())  
            iraungituak.add(produktua);  
    }  
    return iraungituak;  
}
```

**JAVA7**



```
Public List<Produktua> getProduktuIraungituak() {  
    return produktuZerrenda.stream()           //barne iterazioa  
        .filter(p->p.iraungitaDago()) //filtroa   
        .collect(toList()); //zerrenda batean multzokatu  
}
```

**JAVA8**

# Stream eta agregazio operazioak

- ▶ Barne iterazioa:

- Sekuentziala

```
produktuZerrenda.stream()           //sekuentzialki iteratu  
    .filter( p -> p.iraungitaDago()) //filtroa  
    .collect(toList()); //zerrenda batean multzokatu
```

- Paraleloa

```
produktuZerrenda.parallelStream() //paraleloan iteratu  
    .filter( p -> p.iraungitaDago()) //filtroa  
    .collect(toList()); //zerrenda batean multzokatu
```

# 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
<code>filter</code>	<code>Predicate&lt;T&gt;</code>	<code>Stream&lt;T&gt;</code>	Predikatua betetzen duten elementuen fluxua bueltatu.
<code>map</code>	<code>Function&lt;T, R&gt;</code>	<code>Stream&lt;R&gt;</code>	Fluxuko elementu bakoitzari funtzio bat aplikatu, eta emaitza fluxu berri batean bueltatu. Tipo primitiboentzat aldaerak daude ( <code>mapToInt</code> edo <code>mapToDouble</code> )
<code>sorted</code>	<code>Comparator&lt;T&gt;</code>	<code>Stream&lt;T&gt;</code>	Fluxu bateko elementuak baldintza batzuen arabera ordenatu eta emaitza fluxu berri batean bueltatu.
<code>distinct</code>		<code>Stream&lt;T&gt;</code>	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
<code>forEach</code>	<code>Consumer&lt;T&gt;</code>	<code>void</code>	Fluxuko elementu bakoitza kontsumitu, definitutako lambda aplikatuz.
<code>count</code>		<code>long</code>	Fluxuko elementu kopurua bueltatu.
<code>collect</code>	<code>Collector&lt;T, A, R&gt;</code>	<code>R</code>	Fluxua erreduzitu zerrenda mapa edo balio oso bat sortzeko, definitutako rekolekzio metodoaren arabera.
<code>anyMatch</code>	<code>Predicate&lt;T&gt;</code>	<code>boolean</code>	Fluxuko elementuetako batek predikatua betetzen badu, <code>true</code> bueltatu.
<code>allMatch</code>	<code>Predicate&lt;T&gt;</code>	<code>boolean</code>	Fluxuko elementu orok predikatua betetzen badute, <code>true</code> bueltatu.

# Stream eta agregazio operazioak

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

OP	Arg.	Buelta	Helburua
<code>sum</code>		<code>int</code> edo <code>double</code>	Fluxuko elementuen batuketa bueltatu.
<code>average</code>		<code>OptionalDouble</code>	Fluxuko elementuen batzbestekoa bueltatu.
<code>summaryStatistics</code>		<code>IntSummaryStatistics</code> , <code>DoubleSummaryStatistics</code>	Fluxuko elementuen estatistikak bueltatzen ditu



# Stream eta agregazio operazioak

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

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

- Bilduma 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 batzbestekoa?
  - Ezein elementuk bilaketa irizpiderik bete ezean, zer bueltatu?
- **Optional<T>** : Balio bat enkapsulatzeko datu mota, existitzen baldin bada.
  - Metodoak ditu:
    - hutsik dagoen jakiteko: `isPresent`
    - balioa eskatzeko: `get`
    - defektuzko balioa hutsik badago: `orElse`
  - Tipo primitiboentzako implementazioak (`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