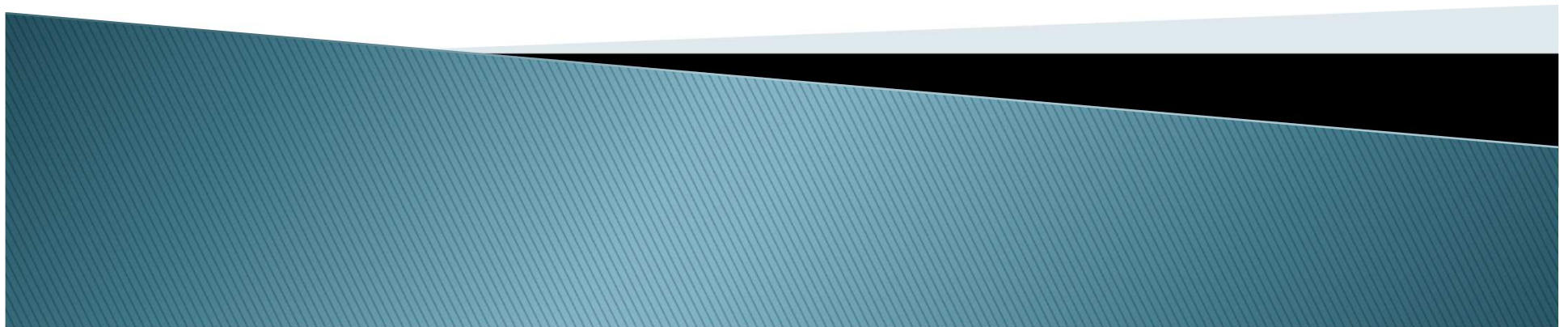


Salbuespenen Tratamendua

SOFTWARE INGENIARITZA



Edukiak

1. Sarrera
2. Salbuespen motak
 - Javan aurredefinitutako salbuespenak
 - Programatzaileak definitutako salbuespenak
3. Salbuespenen kudeaketa Javan
4. Adibideak



Sarrera

- ▶ **Helburua:** sendotasuna eta fidagarritasuna
 - Programazio defentsiboa → erabilera okerrak aurreikusi
 - Hw edo Sw akats baten ondoren programak ez luke gelditu behar
- ▶ **Salbuespena:** programa batek porrot egitea lortu dezakeen exekuzio garaiko gertaera



Adibidea

```
Programa Nagusia{  
...  
    irakurriFitxategia("Froga1.txt")  
}
```

```
Metodoa irakurriFitxategia(pFitxategilzena: String) {  
    lerroa: String  
    ireki(pFitxategilzena)  
    while ez_bukaera(pFitxategilzena){  
        lerroa = irakurriLerroa(pFitxategilzena);  
        tratatu(lerroa);  
    }  
    itxi(pFitxategilzena)  
}
```

Fitxategia existitzen ez bada?



SALBUESPENA

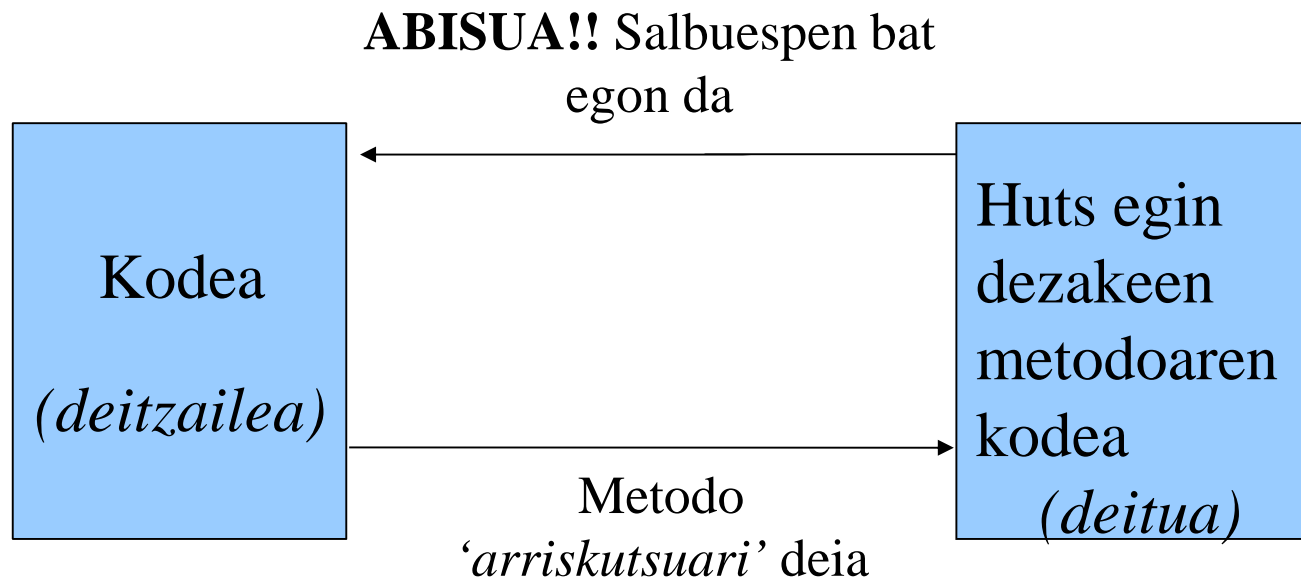
Sarrera

- ▶ Tratamendua:

1. Porrota onartu eta deitu duen programari edo metodoari abisatu
2. Egoera aldatu eta berriz saiatu



Sarrera



Salbuespen Motak

- ▶ Programazio lengoaian aurredefinitutako salbuespenak
 - Ez dira definitu behar (jada definituta daude)
 - Sistemak automatikoki aktibatzen (jaurtitzen) ditu
 - Programatzaileak ere jaurti ditzake
 - Programatzailea arduratzen da bere kudeaketaz
- ▶ Programatzaileak definitutako salbuespenak
 - Programatzaileak definitu, aktibatu eta kudetazen dituen salbuespenak dira



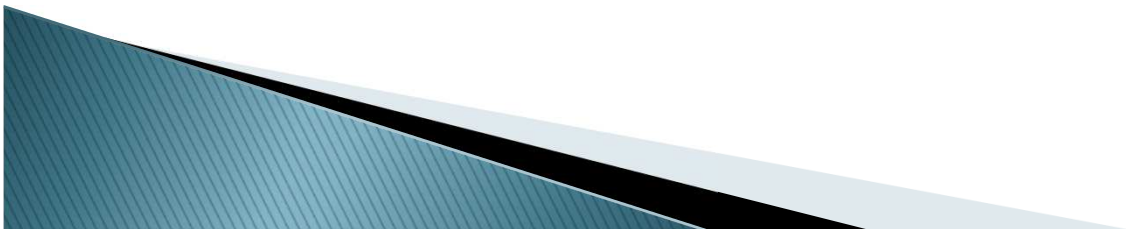
Salbuespenak Javan: Exception klasea

- ▶ Salbuespenak Java klaseak dira
(<https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html>)
- ▶ Oso klase sinplea da
- ▶ Bi konstruktore ditu (beste batzuren artean)
 - Exception()
 - Exception(String mezua)
- ▶ Mezua lortu
 - `excepcion.getMessage();`
- ▶ Aztarna erakutsi
 - `excepcion.printStackTrace();`



Salbuespenen Kudeaketa Javan (I)

- ▶ Zein kode bloketan gerta daitekeen salbuespena adierazi
 - try blokea
- ▶ try blokean gerta daitezkeen salbuespenak tratatzeko beharrezkoa den kodea idatzi
 - Salbuespen kudeatzailea
 - catch bloke sekuentzia
- ▶ try bloke bakoitzaren ondoren gutxienez catch sententzia bat
- ▶ catch bloke bat baino lehen try bloke bat egon behar da



Salbuespenen Kudeaketa Javan (II)

```
try {  
    // akatsa eman dezakeen kodea  
}  
  
catch (Exception e) {  
    // salbuespenaren tratamendua  
}
```

Salbuespena
harrapatu

catch-ean adierazitako
salbuespena gertatzen
bada bakarrik
exekutatuko da

Salbuespenen Kudeaketa Javan (III)

- ▶ try bloke baten barruan salbuespen bat aktibatzen bada kontrola bere kudeatzaileari pasatzen zaio eta try blokearen barruan gelditzen den kodea ez da exekutatu
- catch blokeen artean salbuespenari dagokiona bilatzen da
- Dagokion catch blokea aurkitzen bada, bere kodea exekutatu ondoren try blokearen bukaerara salto egiten da
- Kodearen ordenean bilatzen da → catch sententziak espezifikotasun haundienetik txikienera programatu behar dira
- ▶ try blokean salbuespenik ez badago ez dira catch blokeko sententziak exekutatzen



Salbuespenen Kudeaketa Javan (IV)

- Metodo *deitzaile* batek beste metodo *deituak* jaurtitzen duena harrapatuko du (catch)
- Tratatu gabeko salbuespen bat *deitzailera* hedatuko da (throws)
- Hedatzen duen metodoak hala adierazi beharko du

Adibidea

```
public void metodoArriskutsua() throws Exception {  
    // ....  
    if (zerbaitGaizkiDoa ) {  
        throw new Salbuespena();  
    }  
}
```

throws Exception

Hedatzen da

Deitua

Salbuespena jaurtitzen du

```
public void proba() {
```

Deitzailea

```
    try {  
        obj1.metodoArriskutsua();  
    }
```

*Salbuespena jaurti edo hedatu
dezakeen metodo bat
exekutatzen saiatzen da*

```
    catch (Salbuespena s) {  
        System.out.println("Zorte txarra, ez du funtzionatu");  
        ex.printStackTrace();  
    }  
}
```

*Salpuespena harrapatu
eta tratatu*

Salbuespenen Kudeaketa Javan (IV)

- ▶ Salbuespen bat “hedatuta” iristen bada metodo deitailera, kontrola bere salbuespen kudeatzaileari pasatzen zaio
- ▶ Salbuespena programa nagusira iristen bada kontrola bere kudeatzaileari pasatzen zaio ere bai
 - Aurrikusita badago tratu egiten da eta programa “normal” bukatzen da
 - Ez badago, exekuzio errore bat gertatzen da



Aurredefinitutako Salbuespenak

- ▶ `java.lang.ArithmeticException`
- ▶ `java.lang.ArrayStoreException`
- ▶ `java.lang.IndexOutOfBoundsException`
 - `java.lang.ArrayIndexOutOfBoundsException`
 - `java.lang.StringIndexOutOfBoundsException`
- ▶ `java.lang.IllegalArgumentException`
 - `java.lang.NumberFormatException`
 - `java.lang.IllegalThreadStateException`
- ▶ `java.lang.NegativeArraySizeException`
- ▶ `java.lang.NullPointerException`
- ▶ `java.lang.InterruptedExcep`

Exception-en azpiklaseak Javan

- ▶ `java.io.IOException`
 - `java.io.EOFException`
 - `java.io.FileNotFoundException`
 - `java.io.InterruptedIOException`
- ▶ `java.net.MalformedURLException`



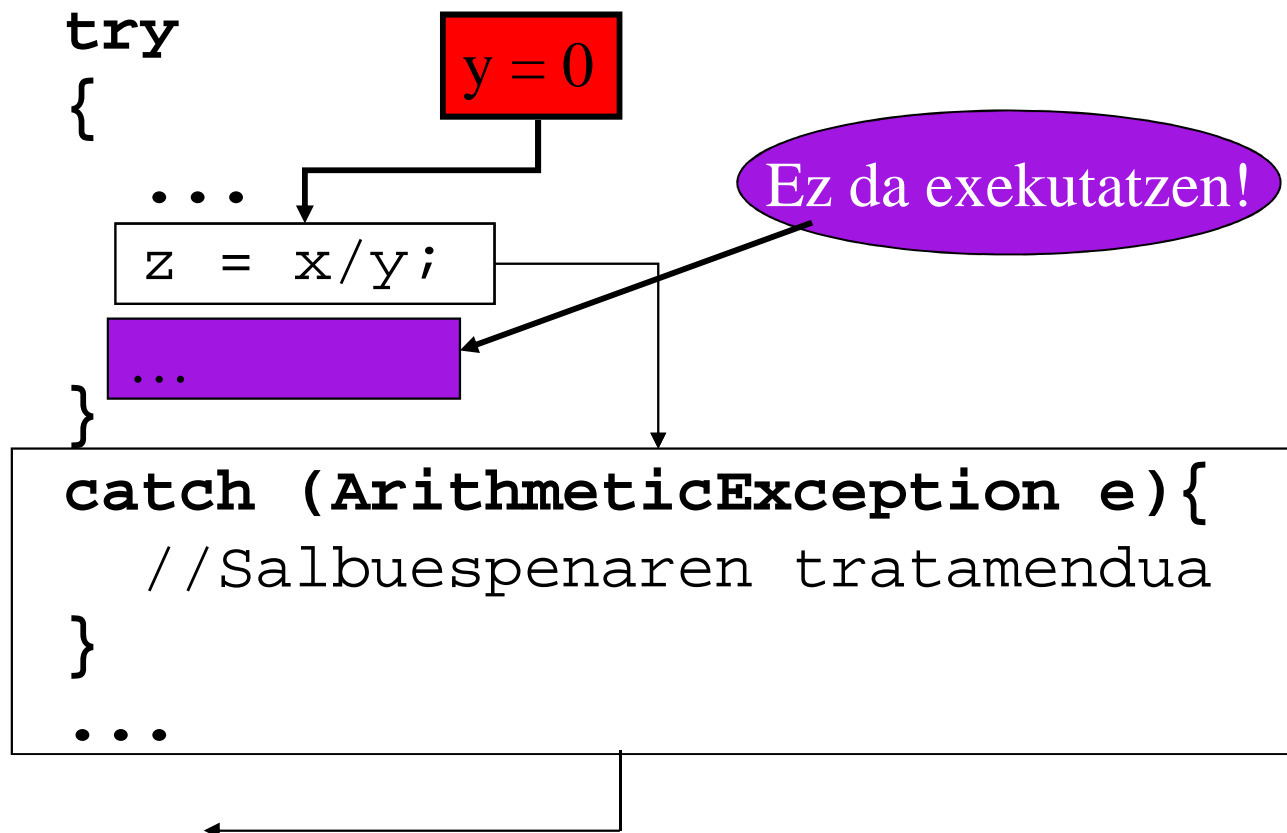
Adibidea

```
public double zatitu () {  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */  
    try {  
        return (x/y);  
    }  
    catch (ArithmeticException e) {  
        //Salbuespena tratatu  
    }  
}
```

y=0 → ArithmeticException
AKATSA SALBUESPEN BEZALA TRATATU



Adibidea



Adibidea

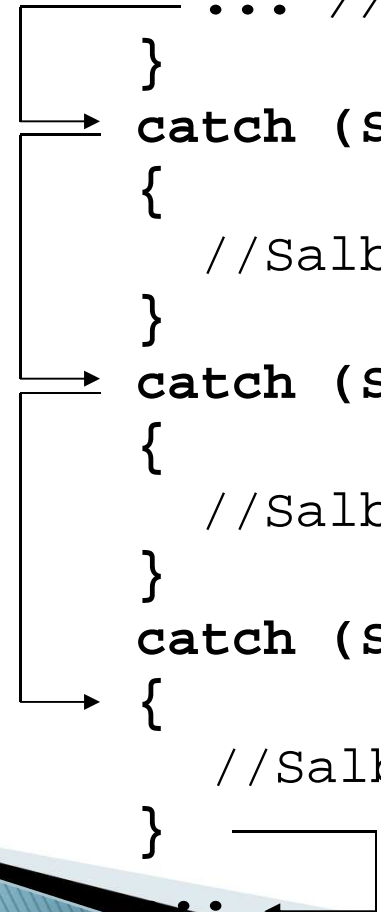
```
public double zatitu () throws ArithmeticException{  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */  
  
    return (x/y);  
  
}
```

y=0 → ArithmeticException
AKATSA SALBUESPEN BEZALA TRATATU



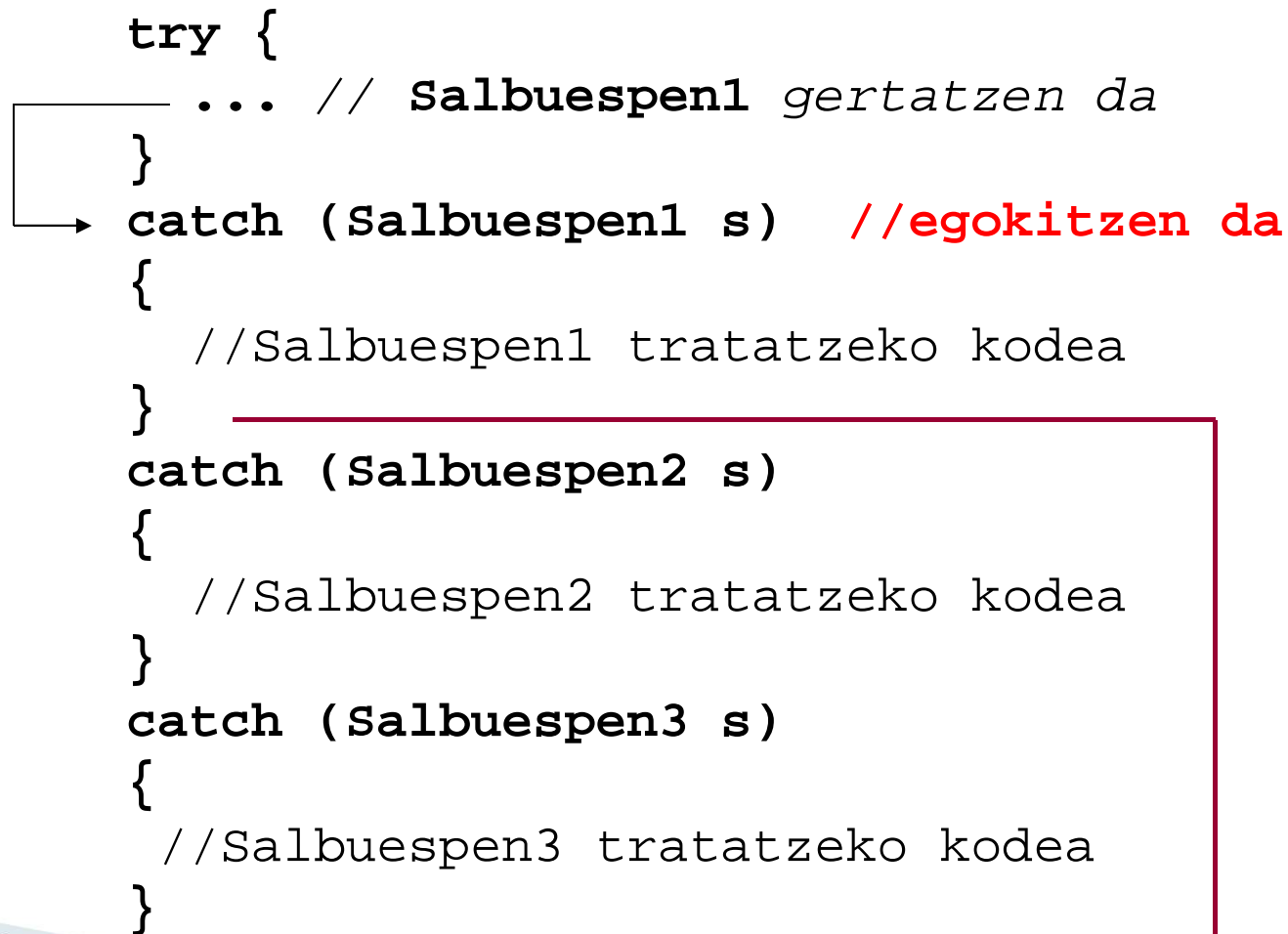
Adibidea

```
try{  
    ... //Salbuespen3 gertatzen da  
}  
→ catch (Salbuespen1 s) //ez da egokitzen  
  {  
    //Salbuespen1 tratatzeko kodea  
  }  
→ catch (Salbuespen2 s) //ez da egokitzen  
  {  
    //Salbuespen2 tratatzeko kodea  
  }  
  catch (Salbuespen3 s) //egokitzen da  
  {  
    //Salbuespen3 tratatzeko kodea  
  }  
  ...
```



Adibidea

```
try {  
    ... // Salbuespen1 gertatzen da  
}  
→ catch (Salbuespen1 s) //egokitzen da  
{  
    //Salbuespen1 tratatzeko kodea  
}  
catch (Salbuespen2 s)  
{  
    //Salbuespen2 tratatzeko kodea  
}  
catch (Salbuespen3 s)  
{  
    //Salbuespen3 tratatzeko kodea  
}
```



Adibidea

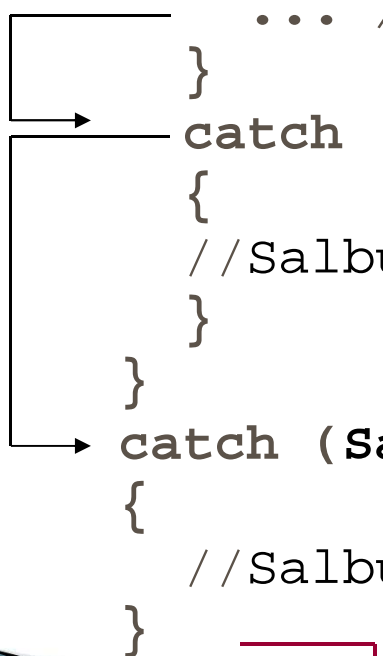
```
try {  
    ... // Salbuespen8 gertatzen da  
}  
→ catch (Salbuespen1 s) //ez da egokitzen  
    {  
        //Salbuespen1 tratatzeko kodea  
    }  
→ catch (Salbuespen2 s) //ez da egokitzen  
    {  
        //Salbuespen2 tratatzeko kodea  
    }  
→ //Salbuespena tratatu gabe gelditu da
```





```
//Metodo deitzailera hedatu behar da  
... //Metodoaren gainerakoa ez da exekutatzen
```


Adibidea

```
try {  
    ...  
    try {  
        ... //Salbuespen2 gertatzen da  
    }  
    catch (Salbuespen1 s) //ez da egokitzen  
    {  
        //Salbuespen1 tratatzeko kodea  
    }  
    catch (Salbuespen2 s) //egokitzen da  
    {  
        //Salbuespen2 tratatzeko kodea  
    }  
}
```



Adibidea

```
try {  
    ...  
    try {  
        ... // Salbuespen1 gertatzen da  
    }  
    catch (Salbuespen1 s) //egokitzen da  
    {  
        //Salbuespen1 tratatzeko kodea  
    }  
    ... // Salbuespen2 gertatzen da  
}   
catch (Salbuespen2 s) //egokitzen da  
{  
    //Salbuespen2 tratatzeko kodea  
}  
... 
```



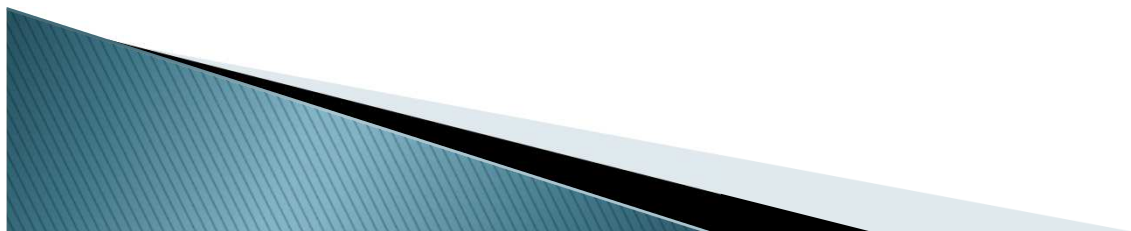
Finally klausula

- ▶ `finally` beti egikaritu behar diren sententzientzat erabiltzen da
 - Salbuespenik egon **ez** bada, `try` blokearen ondoren exekutatzen da
 - Salbuespenik egon bada, egokitzen zaion `catch`-aren ondoren exekutatzen da. Egokitzen zaion `catch`-ik ez badago salbuespena aktibatu ondoren exekutatzen da
 - Beti exekutatzen da, berdin da `try` edo `catch`-etan `return` edo `break` sententziak egotea



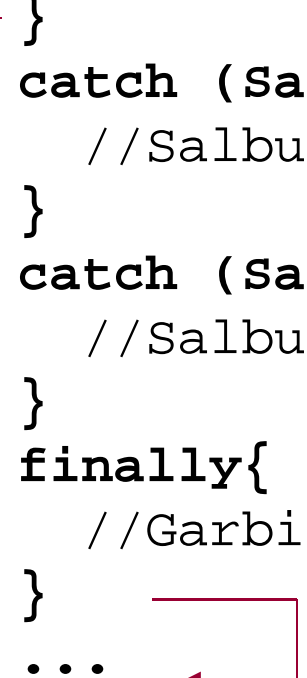
Finally klausula

- ▶ Garbiketa kodea: exekuzioa jarraitu ahal izateko egoera egoki bat berreskuratzen du, nahiz eta salbuespena aktibatuta egon
- ▶ Try bloke bakoitzeko finally klausula bakarra egon daiteke



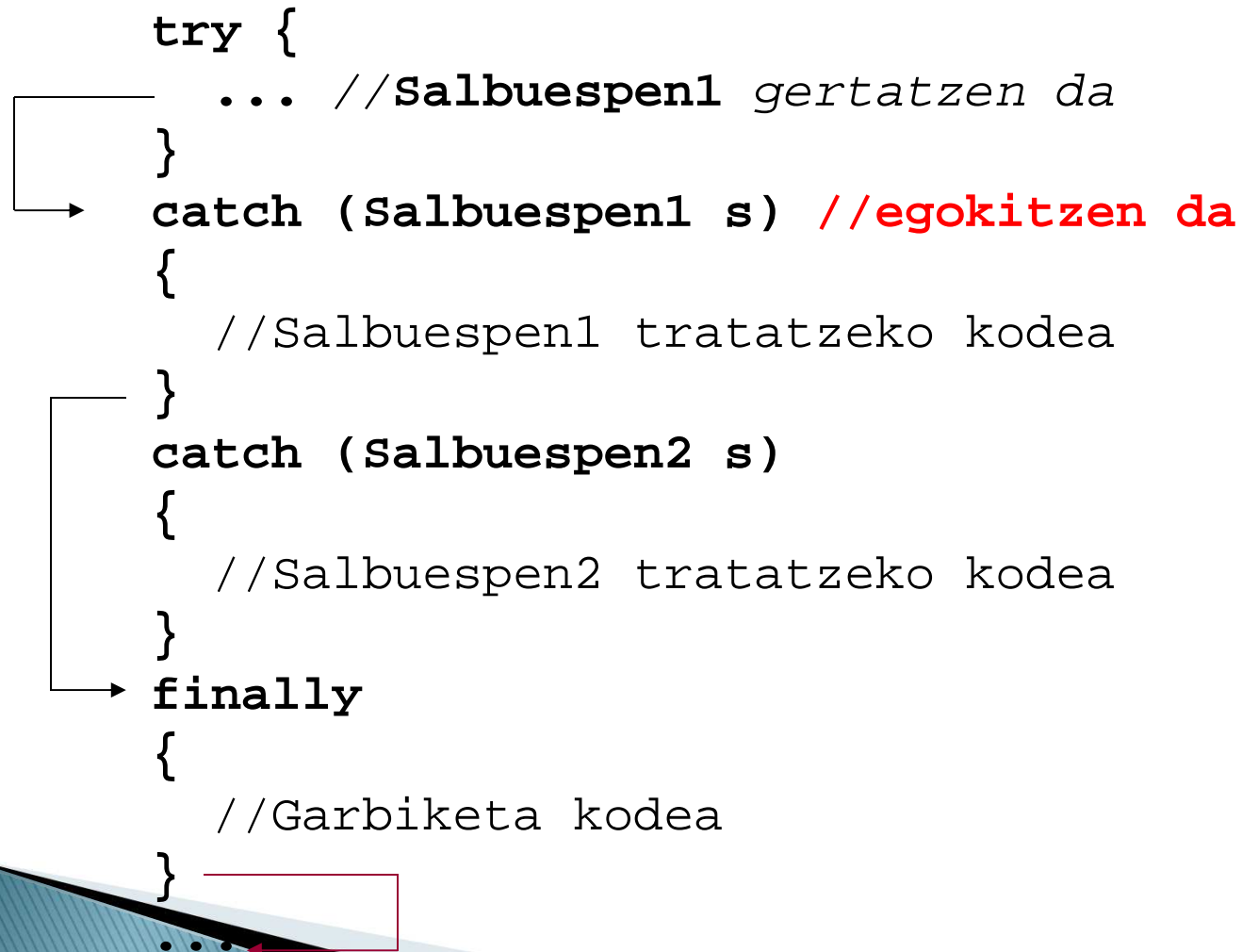
Adibidea

```
try {  
    ... //Salbuespenik gabe bukatzen da  
}  
catch (Salbuespen1 s){  
    //Salbuespen1 tratatzeko kodea  
}  
catch (Salbuespen2 s){  
    //Salbuespen2 tratatzeko kodea  
}  
finally{  
    //Garbiketa kodea  
}  
... 
```



Adibidea

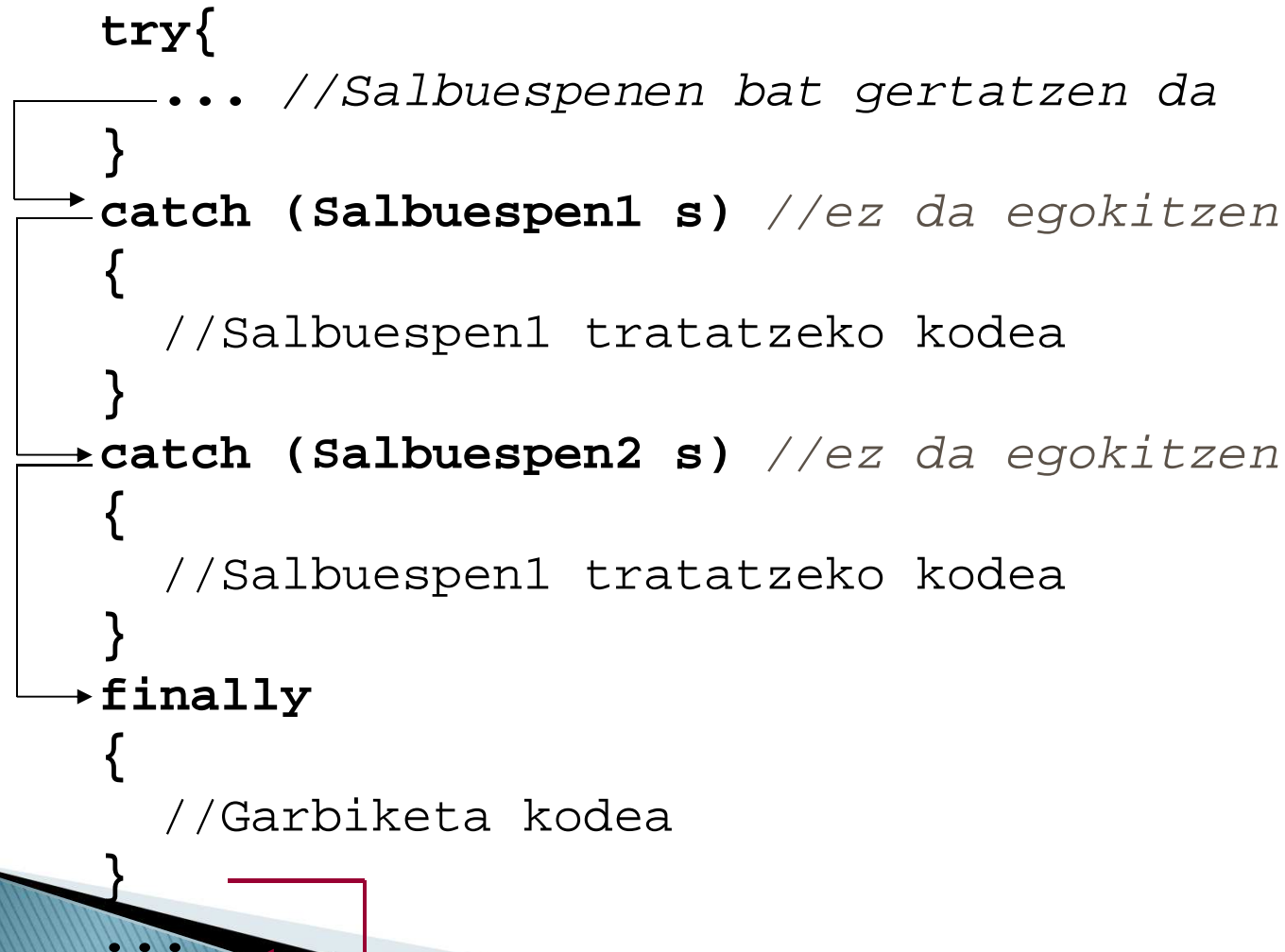
```
try {  
    ... //Salbuespen1 gertatzen da  
}  
→ catch (Salbuespen1 s) //egokitzen da  
{  
    //Salbuespen1 tratatzeko kodea  
}  
→ catch (Salbuespen2 s)  
{  
    //Salbuespen2 tratatzeko kodea  
}  
→ finally  
{  
    //Garbiketa kodea  
}  
→ ...
```



The diagram illustrates the execution flow of a try-catch-finally block. A black arrow points from the opening curly brace of the try block to the catch block, indicating that if an exception of type Salbuespen1 occurs, the code in the catch block will execute. Another black arrow points from the opening curly brace of the catch block to the finally block, indicating that the finally block will execute regardless of whether an exception was caught. A red arrow points from the closing curly brace of the finally block to the ellipsis (...), indicating that the code following the try-catch-finally block will execute after the finally block completes.

Adibidea

```
try{  
    ... //Salbuespenen bat gertatzen da  
}  
→ catch (Salbuespen1 s) //ez da egokitzen  
    {  
        //Salbuespen1 tratatzeko kodea  
    }  
→ catch (Salbuespen2 s) //ez da egokitzen  
    {  
        //Salbuespen1 tratatzeko kodea  
    }  
→ finally  
    {  
        //Garbiketa kodea  
    }  
    ...
```



The diagram illustrates the execution flow of a try-catch-finally block. It shows a sequence of code lines with arrows indicating the flow of execution. The flow starts at the beginning of the try block, goes through the first catch block, then the second catch block, then the finally block, and finally to the ellipsis. A red arrow points from the end of the finally block to the ellipsis, indicating the flow of execution after the finally block.

Salbuespenen Tratamendua

- ▶ Porrota onartu eta metodo deitzaileari abisatu, programa egoera egonkor batetan utziz
- ▶ Nola edo hala salbuespena sortu duen egoera aldatu eta metodoa berriz exekutatzen saiatu
 - Sententzia hurrengo elementuarekin exekutatzen saiatu (fitxategia, taula...)
 - Beste bide bat saiatu
 - Akatsa sortu duena konpondu
 - Itxaron eta berriz saiatu



Adibidea

Porrota onartu eta metodo deitzaileari abisatu, programa egoera egonkor batetan utziz

```
public double zatitu() throws ArithmeticException {  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */  
    try  
        return (x/y);  
    catch (ArithmeticException e) {  
        System.out.println("Ezin da zatitu")  
        throw (new ArithmeticException());  
    }  
}
```

y=0 → ArithmeticException

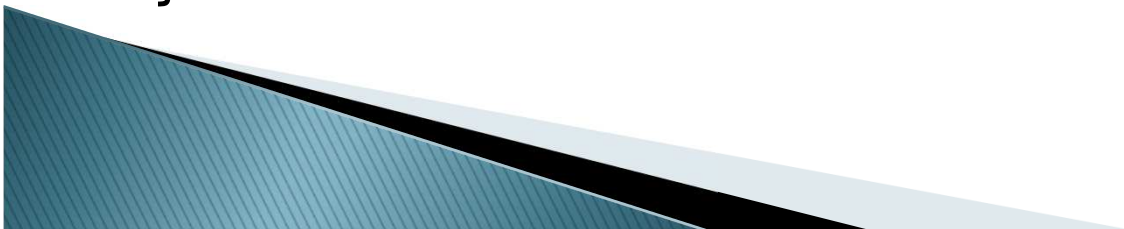
*Metodo deitzaileari
abisatzen dio, salbuespena
pasatuz*

Adibidea

Akatsa sortu duena konpondu

```
public double zatitu () {  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */  
    try  
        return (x/y);  
    catch{  
        y = 1;  
        return (x/y);  
    }  
}
```

y=0 → ArithmeticException

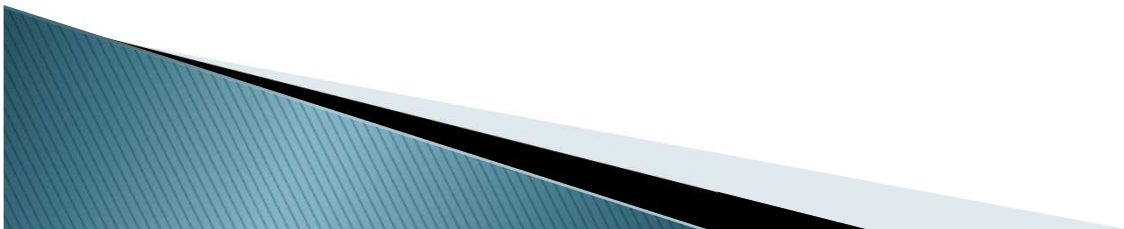


Adibidea

Metodoa berriro exekutatzen saiatu

```
public double zatitu () {  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */  
    try  
        return (x/y);  
    catch{  
        zatitu();  
    }  
}
```

y=0 → ArithmeticException

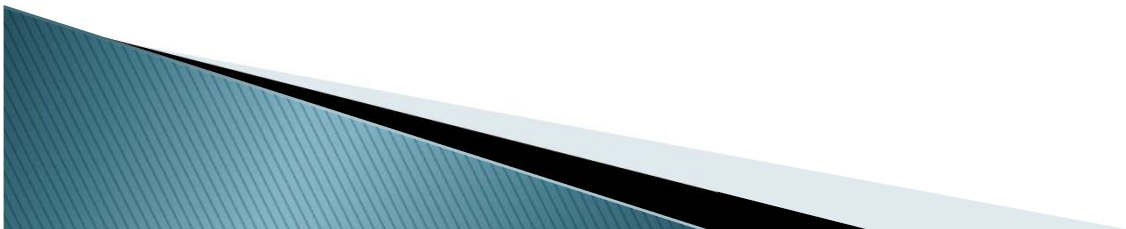


Programatzaileak definitutako salbuespenak

- ▶ Zenbait kasutan programatzaileak bere salbuespen espezifikoak definitu nahiko ditu

throw salbuespenInstantzia

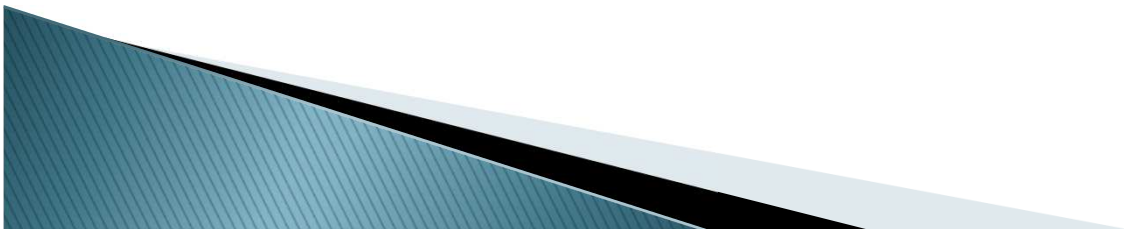
- ▶ Programatzaileak noiz aktibatzen diren kontrolatu beharko du
- ▶ Aurredefinitutakoak bezala tratatzen dira



Programatzaileak definitutako salbuespenak

Programatzailearen salbuespen baten definizioa:

```
class ZatitzaileBikoitia extends Exception {  
    public ZatitzaileBikoitia(){  
        super();  
    }  
  
    public ZatitzaileBikoitia(String s){  
        super(s);  
    }  
}
```



Adibidea

```
public double zatitu(int x, int y) throws
    ZatitzaileBikoitia { HEDATU
    if (y % 2 == 0)
        throw new ZatitzaileBikoitia ("Zenbaki bikoiti
        baten arteko zatiketa");
    } JAURTI
```



Adibidea

```
public static void main(String[] args) {  
    double emaitza;  
    System.out.println("Bi zenbaki sartu: ");  
    /* Zenbakiak jaso eta x, y aldagaietan gorde */
```

```
try
```

```
    emaitza=zatitu(x, y);
```

SAIATU

```
catch (ZatitzaileBikoitia zb){  
    y = y + 1;  
    emaitza = zatitu(x,y);  
}
```

HARRAPATU ETA TRATATU

```
    ...  
}
```



Programatzaileak definitutako salbuespenak

- ▶ Salbuespenak sortzeko jarraitu beharreko pausoak:
 1. Soluzio ideala planteatu
 2. Salbuespen egoerak identifikatu
 3. Salbuespen egoeren tratamedua programatu
 4. Aplikazioa/integrazioa

