



Unió Europea

Fons Social Europeu

L'FSE inverteix en el teu futur



GENERALITAT
VALENCIANA



UF10 - EXCEPCIONS

- Teoria -

PROGRAMACIÓ
CFGs DAM

Autor:

Àngel Olmos Giner

segons el material de Carlos Cacho i Raquel Torres

a.olmosginer@edu.gva.es

2022/2023

EXCEPCIONS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. LLANÇAR EXCEPCIONS (Throw)
3. MANEJAR EXCEPCIONS (try – catch - finally)
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA
5. DEFINIR EXCEPCIONS PRÒPIES

1. INTRODUCCIÓ



Una excepció és un **error semàntic** que es produeix en temps d'execució

Encara que un codi siga correcte sintàcticament (és codi Java vàlid i pot compilar-se), és possible que durant la seua execució es produïsquen errors inesperats, com per exemple ... (?)

- Dividir per zero
- Intentar accedir a una posició d'un array fora dels seus límits
- Al cridar al *nextInt()* d'un *Scanner*, l'usuari no introdueix un valor sencer
- Intentar accedir a un fitxer que no existeix o que està en un disc dur corrupte
- Etc...

```
Output - JavaApplication126 (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at javaapplication126.JavaApplication126.main(JavaApplication126.java:28)
C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run.xml:111: The following error occurred while executing this
C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

1. INTRODUCCIÓ



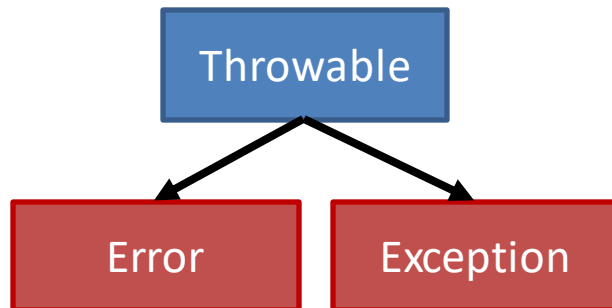
1. INTRODUCCIÓ



1. Quan això ocorre, la JVM **crea un objecte de la classe *Exception*** (les excepcions en Java són TAMBÉ objectes)
2. ... i es notifica al sistema d'execució. Es diu que s'ha llançat una excepció → "*Throwing Exception*"



- Existeixen també els errors interns que són objectes de la classe *Error* que no estudiarem
- Totes dues classes *Error* i *Exception* són classes derivades de la classe base *Throwable*



1. INTRODUCCIÓ



- Un mètode es diu que és capaç de tractar una excepció ("**Catch Exception**") si ha previst l'error que s'ha produït i les operacions a realitzar per a "recuperar" el programa d'aquest estat d'error
- No és suficient capturar l'excepció. Si l'error no es tracta tan sols aconseguirem que el programa no es pare, però l'error pot provocar que les dades o l'execució no siguin correctes



¿SABIAS QUE?

Puedes saltar sin paracaídas de un avión, pero solo una vez

1. INTRODUCCIÓ



- Un mètode es diu que és capaç de tractar una excepció ("**Catch Exception**") si ha previst l'error que s'ha produït i les operacions a realitzar per a "recuperar" el programa d'aquest estat d'error
- No és suficient capturar l'excepció. Si l'error no es tracta tan sols aconseguirem que el programa no es pare, però l'error pot provocar que les dades o l'execució no siguin correctes

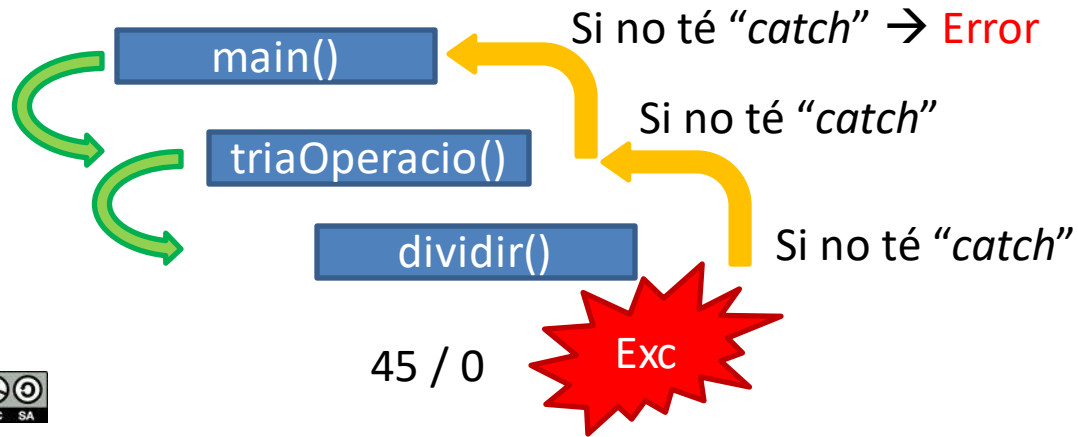
Exemple:

- Volem demanar un *double* per teclat (sc.nextDouble()) per a fer una multiplicació amb una variable a = 3.5
- L'usuari **introdueix un string** en lloc d'un nombre real
- Això provoca una excepció del tipus "**InputMismatchException**"
- Si el programa pot capturar l'excepció, però no la tracta (no intenta corregir l'error):
 1. El programa **no fallarà** i continuarà
 2. Però el **resultat** de la multiplicació que volíem fer serà **incorrecte** (p.e. no es farà)

1. INTRODUCCIÓ



- En el moment en què és llançada una excepció, la JVM **recorre la pila de mètodes** cridats a la recerca d'algun que siga capaç de **tractar l'excepció llançada**:
 1. comença examinant el mètode on s'ha produït l'excepció
 2. si aquest mètode no és capaç de tractarla, examina el mètode des del qual es cridar
 3. i aixina successivament fins a arribar a l'últim mètode (??)
- En cas que cap dels mètodes de la pila siga capaç de tractar l'excepció → JVM mostra un missatge d'error i el programa acaba



```
Output - JavaApplication126 (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at javaapplication126.JavaApplication126.main(JavaApplication126.java:12)
    at C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run
    at C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run
BUILD FAILED (total time: 0 seconds)
```


1. INTRODUCCIÓ

Exemples



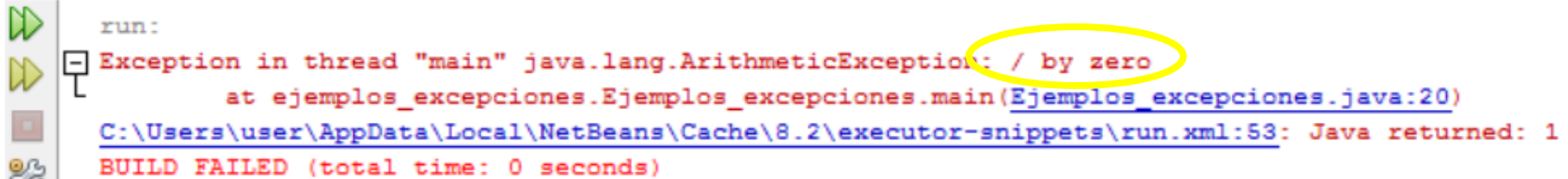
```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int div, x, y;
16
17         x = 3;
18         y = 0;
19
20         div = x / y;
21
22         System.out.println("El resultado es " + div);
23     }
24
25 }
```

Què va a passar?

DIY

1. INTRODUCCIÓ

Exemples



The image shows a screenshot of the NetBeans IDE. On the left, there are icons for running (green play button), debugging (yellow play button), and other IDE functions. The main area displays the following text:

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:20)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

The text `/ by zero` is circled in yellow.

1. El que ha ocorregut és que la JVM ha detectat una condició d'error, la divisió per 0, i ha creat un **objecte de la classe *java.lang.ArithmeticException***
2. Com el mètode on s'ha produït l'excepció no és capaç de tractar-la (**no s'ha implementat cap codi per a gestionar l'excepció**), la JVM busca en el mètode superior
3. ... i com que no hi ha cap altre mètode, finalitza el programa i mostra un missatge d'error amb la informació sobre l'excepció que s'ha produït

1. INTRODUCCIÓ

Exemples



```
12 public class Ejemplos_excepciones {  
13  
14     public static void main(String[] args) {  
15         String cadena = "56s";  
16         int num;  
17  
18         num = Integer.parseInt(cadena);  
19  
20         System.out.println("El número es " + num);  
21     }  
22  
23 }
```

Què va a passar?

DIY

1. INTRODUCCIÓ

Exemples



```
run:
Exception in thread "main" java.lang.NumberFormatException: For input string: "56s"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

1. Com que la cadena no té el format adequat ("56s" no representa un número vàlid), el mètode *Integer.parseInt()* no pot convertir-la a un valor de tipus *int* i **llança l'excepció *NumberFormatException***
2. Com que l'excepció no es tractada, la JVM mostrarà l'error i el programa acaba

1. INTRODUCCIÓ

Exemples



```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int v[] = {1,2,3};
16         int elem;
17
18         elem = v[5];
19
20         System.out.println("El elemento es " + elem);
21
22     }
23
24 }
```

Què va a passar?

DIY

1. INTRODUCCIÓ

Exemples



```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

1. En intentar accedir a una posició que sobrepassa la grandària del vector es produeix una excepció de tipus ***ArrayIndexOutOfBoundsException***
2. Com que novament l'excepció no es tractada, la JVM mostrarà l'error i el programa acaba

EXCEPCIONS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. LLANÇAR EXCEPCIONS (Throw)
3. MANEJAR EXCEPCIONS (try – catch - finally)
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA
5. DEFINIR EXCEPCIONS PRÒPIES



2. LLANÇAR EXCEPCIONS (*Throw*)



Per què llançar excepcions?

- Cal programar el codi de manera que es **llancen excepcions quan s'intente fer una cosa incorrecta** o inesperada → Per exemple, quan els arguments que se li passen a un mètode no són correctes o no compleixen uns certs criteris
- Recordeu que en POO, **una classe és la responsable** de la lògica dels seus objectes: assegurar que les **dades siguin vàlides i controlar què està permès i què no**

Per exemple

- Quan s'instancia un objecte Persona amb un DNI no vàlid, una edat negativa, un compte bancari amb saldo negatiu, etc.
- En aquests casos és convenient que **el constructor llance una excepció** i es gestionen les alternatives

S'encarrega la classe, no el programa principal
o el mètode origen

2. LLANÇAR EXCEPCIONS (*Throw*)

Per què llançar excepcions?



- També és recomanable **llançar excepcions en els *setters*** si el valor no és vàlid, i en **qualsevol altre mètode en el qual s'intente fer una cosa no permesa** o que viole la integritat de l'objecte
- Per exemple → retirar diners d'un compte sense saldo suficient

S'ha de tindre en compte que **les excepcions poden manejar-se i controlar-se sense que el programa es pare** (**next in: apartat 3**)

És a dir, llançar una excepció no implica necessàriament que el programa acabe, és simplement una manera d'avisar d'un error. **Qui crida al mètode és el responsable de manejar l'excepció perquè el programa no es pare**

2. LLANÇAR EXCEPCIONS (*Throw*)

Com llançar una excepció



- S'utilitza la paraula reservada ***throw*** seguida d'un objecte de tipus *Exception* (o alguna de les seues subclasses com *ArithmeticException*, *NumberFormatException*, *ArrayIndexOutOfBoundsException*, etc.)
- Com que **les excepcions són objectes**, deuen instanciar-se amb *new*
- Per tant, podem llançar una excepció genèrica:

`throw new Exception();`



equivalent

```
Exception e = new Exception();  
throw e;
```

Està clar???



2. LLANÇAR EXCEPCIONS (*Throw*)

Com llançar una excepció



- El constructor de ***Exception*** permet un argument ***String*** per a donar detalls sobre el problema
- Si l'excepció no es maneja i el programa es para, el missatge d'error es mostrarà per la consola → molt útil per a depurar programes

throw new Exception("L'edat no pot ser negativa");

- En lloc de llançar excepcions genèriques (*Exception*) també és possible llançar excepcions específiques de Java: *ArrayIndexOutOfBoundsException*, *ArithmeticException*, etc
- A Java **totes les classes d'excepcions hereten de la classe *Exception***

throw new ArithmeticException("No es pot dividir per 0");

- Però ... **és preferible llançar les nostres pròpies excepcions** (apartat 6)



HomeMadeException() 19

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



- És obligatori indicar en la capçalera del mètode que aquest pot llançar excepcions
- Cal afegir la paraula reservada ***throws*** seguida del tipus d'excepció que pot llançar

Exemple

Recupera la classe *Persona* dels exemples de tema anterior i modifica el *setter* de l'edat per a que llance una excepció si es vol assignar una edat negativa

```
if(edat < 0){  
    throw new Exception("L'edat no pot ser negativa");  
}else{  
    this.edat = edat;  
}
```

Que diu *Netbeans*?
Compila el codi?

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



Exemple

Recupera la classe *Persona* dels exemples de tema anterior i modifica el *setter* de l'edat per a que llance una excepció si es vol assignar una edat negativa

```
public void setEdat(int edat) throws Exception{  
    if(edat < 0){  
        throw new Exception("L'edat no pot ser negativa");  
    }else{  
        this.edat = edat;  
    }  
}
```

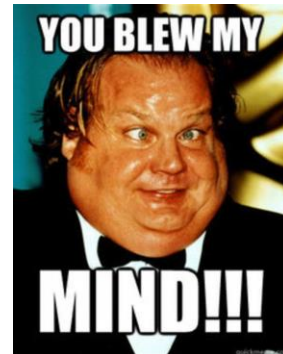
En els mètodes que llancen excepcions, cal afegir en la seua capçalera el tipus d'excepcions que poden llançar

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



```
public void setEdat(int edat) throws Exception{  
    if(edat < 0){  
        throw new Exception("L'edat no pot ser negativa");  
    }else{  
        this.edat = edat;  
    }  
}
```



Un determinat mètode pot llançar excepcions
→ el mètode llança = *throws* (3a persona singular present 's')

Quan es dona la situació incorrecta, es llança l'excepció
→ llançar = *throw* (infinitiu)

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



Està clar???



Todo esta claro,
pero no lo entiendo.

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



Exemple

... modifica ara el programa principal, instancia una nova persona amb una edat negativa i executa el programa

```
Persona p666 = new Persona("00000000Z", "Darth", "Vader", -666);  
System.out.println(p666);
```

Que ha passat?

Nom: Darth, Cognoms: Vader, DNI: 00000000Z, edat: -666

S'ha assignat una edat negativa sense error ... perquè l'excepció està al setter, no al constructor

Si tenim comprovacions als *setters*, és millor cridar al *setter* en el constructor en lloc de fer el *this.X = X*

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode

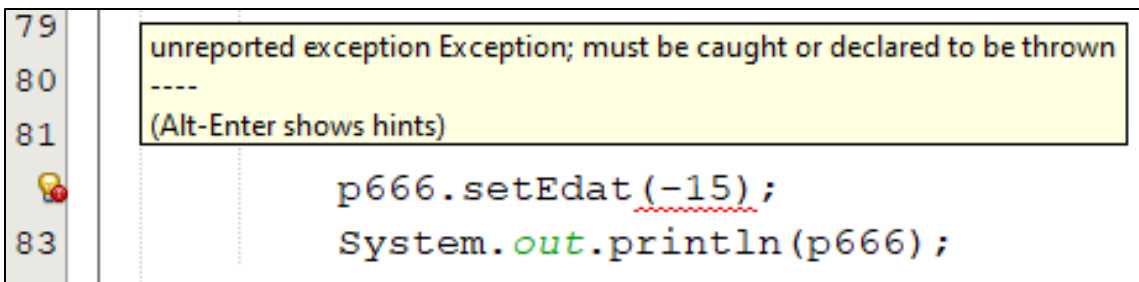


Exemple

... modifica l'edat, utilitzant el *setter*, a un altre valor negatiu i executa el programa

```
p666.setEdat(-15);  
System.out.println(p666);
```

Que ha passat?



Com que `setEdat()` pot crear excepcions, ací:

- o s'agafen i es gestionen (*must be caught*)
- o s'han d'enviar al mètode superior (*declared to be thrown*)


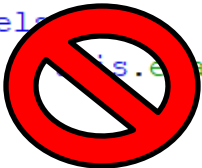
2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



Cal tindre en compte que **quan es llança una excepció es para l'execució del mètode** i es passa l'excepció al mètode que el va cridar

```
public void setEdat(int edat) throws Exception{  
    if(edat < 0){  
        throw new Exception("L'edat no pot ser negativa");  
    }else{  
        this.edat = edat;  
    }  
}
```



```
p666.setEdat(-15);  
System.out.println(p666);
```

Al cridar a *setEdat()* des del *main*, com que *setEdat()* pot llançar una excepció, en la pràctica és com si el *main* llançara una excepció (no directament amb un *throw*, sinó per l'excepció que ens llança *setEdat()*)

2. LLANÇAR EXCEPCIONS (*Throw*)

Indicar l'excepció en la capçalera del mètode



Per tant, per a aquest exemple, cal fer una de les següents coses:

1. Agafar l'excepció (*catch*) i tractar-la → **és el més correcte**. Ho veurem a l'apartat 3
2. Indicar que el *main* també pot llançar una excepció → No és aconsellable que les excepcions arriben de forma incontrolada al *main* i acabe el programa

```
public static void main(String[] args) throws Exception{
```

Executa de nou i ... Que ha passat?

```
Exception in thread "main" java.lang.Exception: L'edat no pot ser negativa
    at javaapplication126.Persona.setEdat(Persona.java:63)
    at javaapplication126.JavaApplication126.main(JavaApplication126.java:83)
C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run.xml:111: The following error
C:\Users\AiA\AppData\Local\NetBeans\Cache\12.0\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

2. LLANÇAR EXCEPCIONS (*Throw*)

Llançant diferents tipus d'excepcions



- Un mètode pot llançar diferents tipus d'excepcions (si ho considerem necessari)
- En tal cas cal especificar tots els tipus possibles en la capçalera, separats per comes

Per exemple, imaginem que el constructor de *Persona* pren com a arguments el dni i l'edat, i volem llançar excepcions diferents segons cada cas

```
public Persona(String dni, int edat) throws InvalidDniException, InvalidEdatException {  
    if (!dni.matches("[0-9]{8}[A-Z]")) {  
        throw new InvalidDniException("DNI no vàlid: " + dni);  
    }  
    if (edat < 0) {  
        throw new InvalidEdatException("Edat no vàlida: " + edat);  
    }  
    this.DNI = dni;  
    this.edat = edat;  
}
```

Que ha passat?



Ja vorem com crear
excepcions pròpies
en l'apartat 5

EXCEPCIONS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. LLANÇAR EXCEPCIONS (Throw)
3. **MANEJAR EXCEPCIONS (try – catch - finally)**
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA
5. DEFINIR EXCEPCIONS PRÒPIES

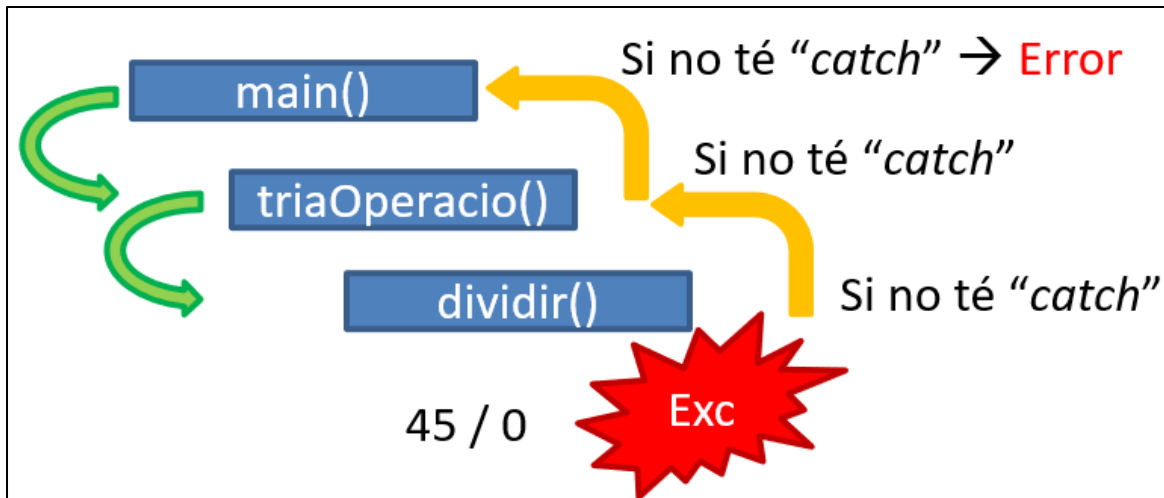


3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



Un **manejador d'excepcions** és una bloc de codi encarregat de tractar les excepcions per a intentar recuperar-se de l'error i **evitar que l'excepció siga llançada descontroladament fins al main i acabe el programa.**



3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



A Java es poden manejar excepcions utilitzant els mecanismes anomenats “manejadors” d'excepcions. Existeixen tres i funcionen conjuntament:

- Bloc ***try*** (intentar): ha d’incloure el codi que podria llançar una excepció
- Bloc ***catch*** (capturar): codi que manejarà (gestionarà) l'excepció si és llançada
- Bloc ***finally*** (finalment): codi que s'executa tant si hi ha excepció com si no

Sempre que s'utilitze un **try** és obligatori utilitzar almenys un **catch**. El **finally** és opcional

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



```
try {  
    // Instruccions que podrien llançar una excepció.  
}  
catch (TipusExcepcio nomVariable) {  
    // Instruccions que s'executen quan 'try' llança una excepció.  
}  
finally {  
    // Instruccions que s'executen tant si hi ha excepció com si no.  
}
```

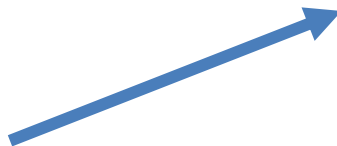
- ❑ El **try** intentarà executar el codi. Si es produeix una excepció s'abandona el bloc sense executar el que queda i se saltarà al bloc *catch*. Si en el *try* no es produeix cap excepció, el bloc *catch* s'ignora
- ❑ El **catch** capturarà les excepcions del tipus *TipusExcepcio*, evitant que siga llançada al mètode superior. Ací haurem d'escriure les instruccions que siguin necessàries per a manejar l'error
- ❑ El bloc **finally** és opcional i s'executarà tant si s'ha llançat una excepció com si no

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadores” d’Excepcions



Codi potencialment “perillós”



```
try  
{
```



```
}  
catch  
{
```



```
}
```

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



Potencialment perillós

Solució



try



catch



Bonus
Track



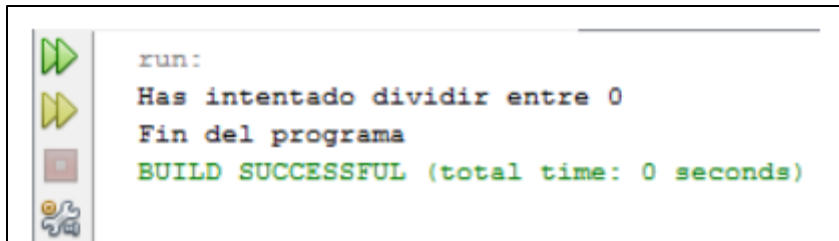
3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



Exemple

1. Implementa en un *main* una divisió entre 2 variables senceres
2. Assigna '0' a la del denominador, executa el programa i analitza l'excepció generada
3. Després modifica el programa per a manejar l'excepció:
 - a) Mou la divisió a un *try*
 - b) Implementa un *catch* que agafe l'excepció i mostre un missatge d'error
 - c) Executa novament el programa per a comprovar que funciona



```
run:
Has intentado dividir entre 0
Fin del programa
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

“Manejadors” d’Excepcions



```
public class Ejemplos_excepciones {  
  
    public static void main(String[] args) {  
        int x = 1, y = 0;  
  
        try  
        {  
            int div = x / y;  
  
            System.out.println("La ejecución no llegará aquí.");  
        }  
        catch(ArithmeticException ex)  
        {  
            System.out.println("Has intentado dividir entre 0");  
        }  
  
        System.out.println("Fin del programa");  
    }  
}
```

- ❑ A l'intentar dividir per zero es llança automàticament una *ArithmeticException*
- ❑ Com això succeeix dins del *try*, l'execució del programa passa al *catch* perquè coincideix amb el tipus d'excepció produïda (*ArithmeticException*)
- ❑ S'executa el codi del *catch* i després el programa continuarà amb normalitat

3. MANEJAR EXCEPCIONS (*try – catch - finally*)



Particularitats de la clàusula *catch*

- El *catch* **sols capturarà excepcions del tipus indicat**. Si es produeix una excepció diferent no la capturarà
- No obstant això, **si que capturarà excepcions heretades**. Per exemple
catch (ArithmeticException e)
capturarà qualsevol tipus d'excepció que herete d'*ArithmeticException*
- El cas més general és ***catch (Exception e)*** que **capturarà tot tipus d'excepcions**
- **Però és millor utilitzar excepcions el més pròximes al tipus d'error previst**, ja que el que es pretén és recuperar el programa d'alguna condició d'error i si "es posen totes les excepcions en el mateix sac", segurament caldrà esbrinar després quina condició d'error es va produir per a poder donar una resposta adequada
- L'objectiu d'una clàusula *catch* és resoldre la condició excepcional perquè el programa pugui continuar com si l'error mai haguera ocorregut

3. MANEJAR EXCEPCIONS (*try – catch - finally*)



Clàusules *catch* múltiples

Es poden especificar diverses clàusules *catch*, tantes com vulguem, perquè cadascuna capture un tipus diferent d'excepció

```
try {  
    // instruccions que poden produir diferents tipus d'Excepcions  
}  
catch (TipusExcepcio1 e1) {  
    // instruccions per a manejar un TipusExcepcio1  
}  
catch (TipusExcepcio2 e2) {  
    // instruccions per a manejar un TipusExcepcio2  
}  
...  
}  
catch (TipusExcepcioN eN) {  
    // instruccions per a manejar un TipusExcepcioN  
}  
finally { // opcional  
    // instruccions que s'executaran tant si hi ha excepció com si no  
}
```

3. MANEJAR EXCEPCIONS (*try – catch - finally*)



Clàusules *catch* múltiples

- Quan es llança una excepció dins del *try*:
 1. es comprova cada sentència *catch* per ordre
 2. s'executa la primera el tipus de la qual coincideix amb l'excepció llançada
 3. els altres blocs *catch* seran ignorats
- Després s'executarà el bloc *finally* (si s'ha definit) i el programa continuarà la seua execució després del bloc *try-catch-finally*



Si el tipus excepció produïda no coincideix amb cap dels *catch*, llavors l'excepció serà llançada al mètode que ens va cridar.

Si es vol que l'excepció no siga llançada al mètode superior

→ fer un *catch(Exception e){ ... }* al final, que només s'executarà quan l'excepció no coincideix amb cap dels altres *catch*

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

Clàusules *catch* múltiples



Exemple

Modifica l'exemple anterior de la següent forma:

1. Crea un *array* de sencers i assigna-li 3 valors
2. Dins el *try*, pregunta a l'usuari:
 - a) quins són els valors a dividir i mostra el resultat
 - b) quina posició de l'*array* vol i mostra el contingut per pantalla
3. Modifica el programa per a gestionar excepcions quan:
 - a) Es fa una divisió per '0'
 - b) L'índex triat per l'usuari no siga un sencer
 - c) El sencer seleccionat per l'usuari apunta a una posició fora de l'*array*

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

Clàusules *catch* múltiples



Exemple

```
int x,y,div,pos;
int [] vec = {1,2,3};
Scanner sc = new Scanner(System.in);

try{
    System.out.print("Numerador? ");
    x = sc.nextInt();
    System.out.print("Denominador? ");
    y = sc.nextInt();
    div = x / y;
    System.out.println("La divisió es: " + div);

    System.out.print("Posició de l'array a mostrar? ");
    pos = sc.nextInt();
    System.out.println("El contingut és: " + vec[pos]);
}
catch(ArithmeticException e){
    System.out.println("Divisió per zero: " + e);
}
catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Has triat una posició fora de l'array: " + e);
}
catch(InputMismatchException e){
    System.out.println("Has de triar un número sencer: " + e);
}
System.out.println("Fi del Programa !!!");
```

3. MANEJAR EXCEPCIONS (*try – catch - finally*)

Clàusules *catch* múltiples



Exemple

Poden succeir **X** coses → **quines?**

1. El *try* s'executa sense excepcions, s'ignoren els *catch* i s'imprimeix “Fi del Programa”
2. Es produeix l'excepció de divisió per zero, el flux d'execució salta al 1er *catch*, s'imprimeix el missatge “Divisió por zero” i després “Fi del Programa”
3. Es produeix l'excepció de sobrepassar el vector, el flux d'execució salta al 2on *catch* i s'imprimeix ...
4. Es produeix l'excepció de tipus incorrecte, el flux d'execució salta al 3r *catch* i ...

```
Numerador? 88
Denominador? 5
La divisió es: 17
Posició de l'array a mostrar? RR
Has de triar un número sencer: java.util.InputMismatchException
Fi del Programa !!!
BUILD SUCCESSFUL (total time: 7 seconds)
```

3. MANEJAR EXCEPCIONS (*try – catch - finally*)



L'objecte *Exception*

- ☐ Tota **excepció genera un objecte de la classe *Exception*** (o un més específic que hereta de *Exception*: *ArithmeticException*, ...)
- ☐ Aquest objecte contindrà detalls sobre l'error produït
- ☐ Pot ser interessant **mostrar aquesta informació**:
 - a) perquè la veja l'usuari/a i que sàpia què ha succeït
 - b) o el desenvolupador/a per a depurar i corregir el codi
- ☐ En la clàusula *catch* tenim accés a l'objecte en cas que vulguem utilitzar-lo

3. MANEJAR EXCEPCIONS (*try – catch - finally*)



L'objecte *Exception*

- ❑ Els dos mètodes d'*Exception* més útils són:
 - ***getMessage()*** → Retorna un String amb un text simple sobre l'error
 - ***printStackTrace()*** → És el que més informació proporciona: Indica quin tipus d'excepció s'ha produït, el missatge simple, i també tota la pila de crides (el que fa Java per defecte)
- ❑ Els objectes de tipus *Exception* tenen implementat el mètode *toString()* → és possible imprimir-los directament mitjançant *println()*

```
catch (Exception e){  
  
    // Mostrem el missatge de l'excepció  
    System.err.println("Error: " + e.getMessage());  
  
    // Anem mostrar tota la informació, missatge i pila de crides  
    e.printStackTrace();  
}
```

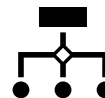
```
// Mostrem el missatge de l'excepció  
System.out.println(e);
```

DIY

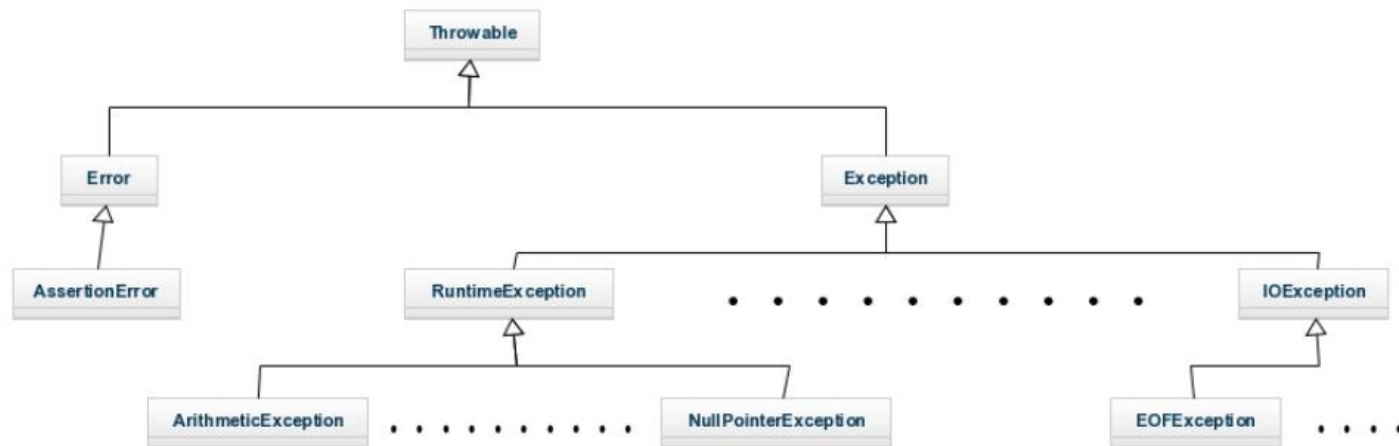
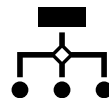
EXCEPCIONS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. LLANÇAR EXCEPCIONS (Throw)
3. MANEJAR EXCEPCIONS (try – catch - finally)
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA
5. DEFINIR EXCEPCIONS PRÒPIES

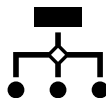


4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA



- ❑ La classe *Exception* hereta de *Throwable* i **totes les excepcions hereten de *Exception***
- ❑ Com *java.lang* és importat de forma implícita en tots els programes, **la major part de les excepcions derivades de *RuntimeException* estan disponibles de manera automàtica**. A més no és necessari incloure-les en cap capçalera de mètode mitjançant *throws*.
- ❑ Les Excepcions poden ser comprovades i no comprovades

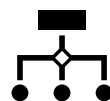
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA



- ❑ **Excepcions comprovades:** aquelles que Java comprova durant la compilació, abans de l'execució del programa

Excepció	Significat
<i>ClassNotFoundException</i>	No s'ha trobat la classe.
<i>CloneNotSupportedException</i>	Intent de duplicat d'un objecte que no implementa la interfície clonable.
<i>IllegalAccessException</i>	S'ha denegat l'accés a una classe.
<i>InstantiationException</i>	Intent de crear un objecte d'una classe abstracta o interfície.
<i>InterruptedException</i>	Fil interromput per un altre fil.
<i>NoSuchFieldException</i>	El camp sol·licitat no existeix.
<i>NoSuchMethodException</i>	El mètode sol·licitat no existeix.

4. JERARQUIA i TIPUS D'EXCEPCIONS JAVA



- ❑ **Excepcions no comprovades:** aquelles que Java no pot comprovar durant la compilació i es produiran durant l'execució del programa

Subclasses de
RuntimeException
no comprovades

Excepció	Significat
<i>AritmeticException</i>	Error aritmètic com a divisió entre zero.
<i>ArrayIndexOutOfBoundsException</i>	Índex de la matriu fora del seu límit.
<i>ArrayStoreException</i>	Assignació a una matriu de tipus incompatible.
<i>ClassCastException</i>	Conversió invàlida.
<i>IllegalArgumentException</i>	Ús invàlid d'un argument en cridar a un mètode.
<i>IllegalMonitorStateException</i>	Operació de monitor invàlida, com esperar un fil no bloquejat.
<i>IllegalStateException</i>	L'entorn o aplicació estan en un estat incorrecte.
<i>IllegalThreadStateException</i>	L'operació sol·licitada és incompatible amb l'estat actual del fil.
<i>IndexOutOfBoundsException</i>	Algun tipus d'índex està fora del seu rang o del seu límit.
<i>NegativeArraySizeException</i>	La matriu té una grandària negativa.
<i>NullPointerException</i>	Ús incorrecte d'una referència NULL.
<i>NumberFormatException</i>	Conversió incorrecta d'una cadena a un format numèric.
<i>SecurityException</i>	Intent de violació de seguretat.
<i>StringIndexOutOfBoundsException</i>	Intent de sobrepassar el límit d'una cadena.
<i>TypeNotPresentException</i>	Tipus no trobat.
<i>UnsupportedOperationException</i>	Operació no admesa.

EXCEPCIONS

ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ
2. LLANÇAR EXCEPCIONS (Throw)
3. MANEJAR EXCEPCIONS (try – catch - finally)
4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA
5. **DEFINIR EXCEPCIONS PRÒPIES**



5. DEFINIR EXCEPCIONS PRÒPIES



- Quan desenvolupem programari, sobretot en desenvolupar les nostres pròpies classes, és habitual que es **vulguem produir excepcions que no estan definides en Java**
- Exemples???

 - Quan un *argin* està fora d'un determinat rang
 - Quan un String no té el format desitjat (DNI, IBAN ...)
 - Quan s'ha superat el nombre màxim d'intents d'escriure un *password*
 - ...

- Per a crear una excepció pròpia **hem de definir una nova classe derivada d'*Exception***
- L'ús d'aquesta nova excepció és el mateix que hem vist

5. DEFINIR EXCEPCIONS PRÒPIES



Exemple

- Crea el mètode *duplicar* que mostrarà el resultat de multiplicar un valor per 2, però només en el cas que el valor siga ≤ 10
- Si és > 10 llançarà una excepció pròpia anomenada *ExcepcioPropia* que agafarà el valor com a argument d'entrada
- L'excepció sobreescriurà el *toString* per a indicar l'error i el valor introduït
- El programa principal haurà d'utilitzar el mètode *duplica* i manejar l'excepció mostrant un missatge explicatiu sobre la captura de l'excepció i l'error

```
Anem a duplicar el valor: 2
Duplicat = 4
Anem a duplicar el valor: 33
*** Excepció capturada: Excepció_Pròpia [el valor 33 no pot ser > 10]
BUILD SUCCESSFUL (total time: 2 seconds)
```

5. DEFINIR EXCEPCIONS PRÒPIES



Exemple

```
public static void duplicar(int valor) throws ExcepcioPropia {  
    System.out.println("Anem a duplicar el valor: " + valor);  
    if (valor > 10) {  
        throw new ExcepcioPropia(valor);  
    }  
    System.out.println("Duplicat = " + valor * 2);  
}
```

Només arriba ací si no
es llança l'excepció

```
class ExcepcioPropia extends Exception{  
    private int valor;  
  
    public ExcepcioPropia(int valor){  
        this.valor = valor;  
    }  
  
    @Override  
    public String toString(){  
        return "Excepció Pròpia [el valor " + valor + " no pot ser > 10]";  
    }  
}
```

Atributs

Co

```
try {  
    duplicar(2);  
    duplicar(33);  
} catch (ExcepcioPropia e) {  
    System.out.println("*** Excepció capturada: " + e);  
}
```

Mètodes

Fes els Exercicis

Autor:

Àngel Olmos Giner

segons el material de Carlos Cacho i Raquel Torres

Llicència:



CC BY-NC-SA 3.0 ES Reconeixement – No Comercial – Compartir Igual (by-nc-sa)

No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. Aquesta és una obra derivada de l'obra original de Carlos Cacho i Raquel Torres