



UNITAT 10. EXCEPCIONS

PROGRAMACIÓ
CFGs DAW

Autors:

Carlos Cacho y Raquel Torres

Revisat per:

Lionel Tarazon - lionel.tarazon@ceedcv.es

Fco. Javier Valero – franciscojavier.valero@ceedcv.es

José Manuel Martí - josemanuel.marti@ceedcv.es

2021/2022

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.

Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per a distingir elements importants dins del contingut. Aquests símbols són:



Important



Atenció



Interessant

ÍNDEX DE CONTINGUT

Introducció	4
Exemple 1	5
Exemple 2	6
Exemple 3	7
LLANÇAR EXCEPCIONS (Throw)	8
Per què llançar excepcions?	8
Com llançar una excepció	8
Indicar l'excepció en la capçalera del mètode	9
Llançant diferents tipus d'excepcions	10
Les excepcions es llancen de mètode a mètode ("la patata caliente")	10
MANEJAR EXCEPCIONS (TRY – CATCH – FINALLY)	11
Manejadors d'excepcions	11
Exemple 4	12
Particularitats de la clàusula catch	12
Clàusules catch múltiples	13
Exemple 5	14
L'objecte Exception	15
JERARQUIA I TIPUS D'EXCEPCIONS JAVA	16
DEFINIR EXCEPCIONS PRÒPIES	18
Exemple 6	18
AGRAÏMENTS	19

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 enter.
- Intentar accedir a un fitxer que no existeix o que està en un disc dur corrupte.
- Etc.

Quan això ocorre, la màquina virtual Java crea un objecte de la classe ***Exception*** (les excepcions en Java són objectes) i es notifica el fet 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***.

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'aqueix 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 siguen correctes.

En el moment en què és llançada una excepció, la màquina virtual Java recorre la pila de cridades a mètodes a la recerca d'algun que siga capaç de tractar la classe d'excepció llançada. Per a això, comença examinant el mètode on s'ha produït l'excepció; si aquest mètode no és capaç de tractar-la, examina el mètode des del qual es va realitzar la cridada al mètode on es va produir l'excepció i així successivament fins a arribar a l'últim d'ells. En cas que cap dels mètodes de la pila siga capaç de tractar l'excepció, la màquina virtual Java mostra un missatge d'error i el programa acaba.

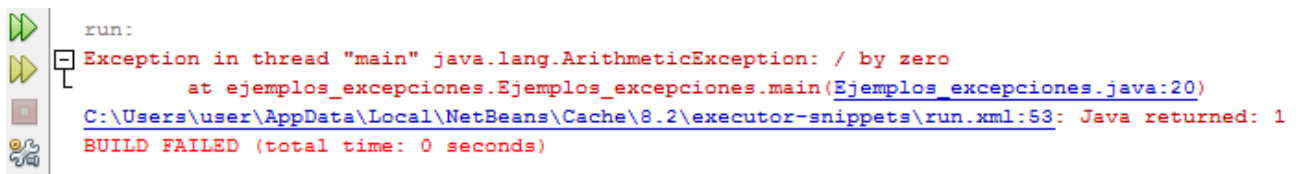
Els programes escrits en Java també poden llançar excepcions explícitament mitjançant la instrucció ***throw***, la qual cosa facilita la devolució d'un "codi d'error" al mètode que va invocar el mètode que va causar l'error.

1.1 Exemple 1

Com a primera trobada amb les excepcions, executarem el següent programa. En ell forçarem una excepció en intentar dividir un número entre 0:

```
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 }
```

Sent l'eixida:



```
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)
```

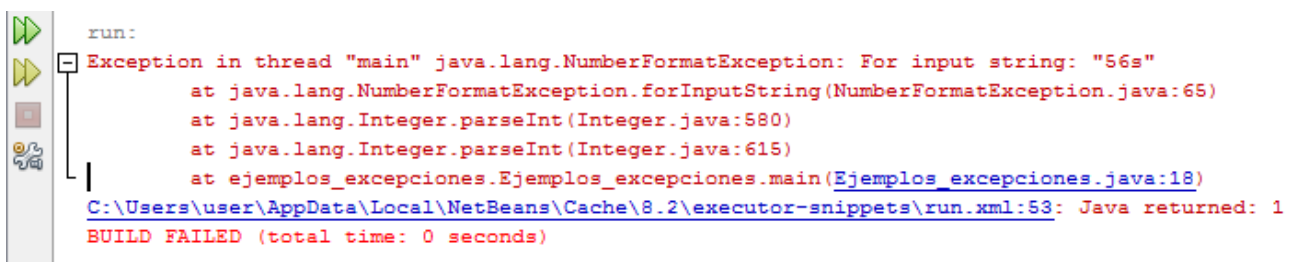
El que ha ocorregut és que la màquina virtual Java (el programa que executa codi Java) ha detectat una condició d'error, la divisió per 0, i ha creat un objecte de la classe *java.lang.ArithmeticException*. Com el mètode on s'ha produït l'excepció no és capaç de tractar-la, la màquina virtual Java finalitza el programa en la línia 20 i mostra un missatge d'error amb la informació sobre l'excepció que s'ha produït.

1.2 Exemple 2

A continuació forçarem una excepció de conversió, per a això intentarem passar a enter una cadena que no sols porta caràcters numèrics:

```
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 }
```

Sent l'eixida:



```
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)
```

Pel fet 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*. La màquina virtual Java finalitza el programa en la línia 18 i mostra per pantalla la informació sobre l'excepció que s'ha produït.

1.3 Exemple 3

En aquest exemple forçarem una excepció de límits del vector, per a això crearem un vector i intentar accedir a una posició que no existeix:

```
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 }
```

Sent l'eixida:

```
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)
```

En intentar accedir a una posició que sobrepassa la grandària del vector es produeix una excepció de tipus *ArrayIndexOutOfBoundsException*. La màquina virtual de java finaliza el programa en la línia 18 i mostra el missatge d'error sobre l'excepció que s'ha produït.

2. LLANÇAR EXCEPCIONS (THROW)

2.1 Per què llançar excepcions?

Un programador pot programar el seu codi de manera que es llancen excepcions quan s'intente fer una cosa incorrecta o inesperada (a vegades és recomanable). Per exemple, quan els arguments que se li passen a un mètode no són correctes o no compleixen uns certs criteris.

Això és habitual en la Programació Orientada a Objectes (POO): Recordeu que una classe ha de ser la responsable de la lògica dels seus objectes: assegurar que les dades siguin vàlides, a més de controlar què està permès i què no està permès fer. **Per exemple si s'instancia una classe amb valors incorrectes** com un objecte Persona amb un DNI no vàlid, una edat negativa, un compte bancari amb saldo negatiu, etc. En aqueixos casos **és convenient que el constructor llance una excepció**.

També, **pot ser apropiat 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** com 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** (en l'apartat 3 s'explica com).

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

2.2 Com llançar una excepció

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

```
throw new Exception();
```

Això és equivalent a primer instanciar l'objecte *Exception* i després llançar-ho:


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

El constructor de *Exception* permet (opcionalment) un argument *String* per a donar detalls sobre el problema. Si l'excepció no es maneja i el programa espera, el missatge d'error es mostrarà per la consola (això és 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 com per exemple *ArrayIndexOutOfBoundsException*, *ArithmeticException*, *NumberFormatException*, etc. A Java totes les classes d'excepcions hereten de Exception.

```
throw new NumberFormatException("...");
```

 En lloc de llançar excepcions pròpies de Java (*ArithmeticException*, *NumberFormatException*, *ArrayIndexOutOfBoundsException*, etc.) **normalmente és preferible llançar excepcions genèriques 'Exception' o millor encara, utilitzar les nostres pròpies excepcions** (apartat 5).

2.3 Indicar l'excepció en la capçalera del mètode

Es obligatori indicar en la capçalera del mètode que pot llançar excepcions. Per a això s'ha d'afegir, a la dreta de la capçalera i abans de les claus, la paraula reservada **throws** seguit de la **mena d'excepció** que pot llançar (si pot llançar diferents tipus d'excepcions deuen indicar-se totes separades per comes).

Per exemple, vegem el **mètode setEdat(int edat)** de la **classe Persona**. Com hem decidit que l'edat d'una persona no pot ser negativa, llançarem una excepció si `edat < 0`.

```
// Setter de l'atribut edat. Ha de ser >= 0
public void setEdat(int edat) throws Exception {
    if (edat < 0)
        throw new Exception("Edat negativa no permesa");
    else
        this.edat = edat;
}
```

Cal tindre en compte que **en llançar una excepció es parará l'execució d'aquest mètode (no s'executarà la resta del codi del mètode) i es llançarà l'excepció al mètode que el va cridar**. Si per exemple des de la funció `main` cridem a `setEdat()`, com pot succeir que `setEdat()` llance una excepció, llavors en la pràctica és possible que el `main` llance una excepció (no directament amb un `throw`, sinó per l'excepció que ens llança `setEdat()`). Per tant, també hem d'especificar en el `main` que es pot llançar una excepció:

```
public static void main(String[] args) throws Exception {
    Persona p = new Persona("44193900L", "Pepito", 27);
    ...
    ...
    p.setEdat(valor); // Pot llançar una excepció
}
```

2.4 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;
}
```



Compte: tingueu en compte que *InvalidEdadException* i *InvalidDniException* no són tipus d'excepcions de Java, sinó excepcions pròpies que nosaltres hauríem definit a mida per al nostre programari (veure apartat 5).

2.5 Les excepcions es llancen de mètode a mètode (“hot potato”)

Observeu que el llançament d'excepcions es produeix recursivament a través de la seqüència de cridades a mètodes. Imaginem que en el mètode A cridem al mètode B, que crida al mètode C, etc. fins a arribar a E.

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ (seqüència de crides de mètodes)

Si el mètode E llança una excepció, aquesta li arribarà a D que al seu temps li la llançarà a C, etc. recorrent el camí invers fins a arribar al mètode inicial A.

$A \leftarrow B \leftarrow C \leftarrow D \leftarrow E$ (seqüència de llançament d'excepcions)

Per tant, com tots aquests mètodes poden acabar llançant una excepció, en les seues capçaleres caldrà incloure el **throws Exception** (o el que corresponga segons el tipus d'excepció).



Compte! No és recomanable deixar que les Excepcions del nostre programari arriben de manera descontrolada fins al main i acaben el programa.

El més ideal és manejar les excepcions com veurem en el següent apartat.

3. MANEJAR EXCEPCIONS (TRY – CATCH – FINALLY)

3.1 Manejadors d'excepcions

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

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



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**.

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

```
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 bloc try intentarà executar el codi. Si es produeix una excepció s'abandona aquest bloc (no s'executaran les altres instruccions del try) i se saltarà al bloc catch. Lògicament, si en el try no es produeix cap excepció el bloc catch s'ignora.

El bloc catch capturarà les excepcions del tipus *TipusExcepcio*, evitant que siga llançada al mètode que ens va cridar. Ací haurem d'escriure les instruccions que siguen necessàries per a manejar l'error. Poden especificar-se diversos blocs catch per a diferents tipus d'excepcions.

El bloc finally és opcional i s'executarà tant si s'ha llançat una excepció com si no.

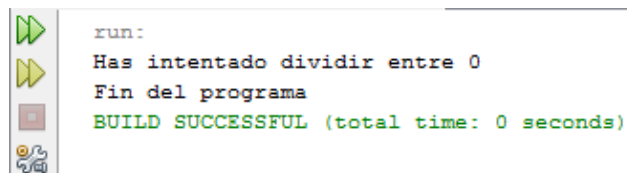
Vegem un exemple senzill.

3.2 Exemple 4

Vamos a produir una excepció i tractar-la fent ús dels manejadors *try-catch*:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int x = 1, y = 0;
16
17         try
18         {
19             int div = x / y;
20
21             System.out.println("La ejecución no llegará aquí.");
22         }
23         catch(ArithmeticException ex)
24         {
25             System.out.println("Has intentado dividir entre 0");
26         }
27
28         System.out.println("Fin del programa");
29     }
30 }
```

Sent l'eixida:



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

Al intenta dividir per zero es llança automàticament una *ArithmeticException*. Com això succeeix dins del bloc *try*, l'execució del programa passa al primer bloc *catch* perquè coincideix amb la mena d'excepció produïda (*ArithmeticException*). S'executarà el codi del bloc *catch* i després el programa continuarà amb normalitat.

3.3 Particularitats de la clàusula *catch*

És important entendre que **el bloc *catch* sols capturarà excepcions del tipus indicat**. Si es produeix una excepció diferent no la capturarà. No obstant això **capturarà excepcions heretades del tipus indicat**. Per exemple, *catch (ArithmeticException e)* capturarà qualsevol tipus d'excepció que herete de *ArithmeticException*. El cas més general és *catch (Exception e)* que capturar tot tipus d'excepcions perquè **a Java totes les excepcions hereten de *Exception***.

No obstant això, és millor utilitzar excepcions el més pròximes a la mena d'error previst, ja que el que es pretén és recuperar el programa d'alguna condició d'error i si "es fiquen 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.

3.4 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  
}
```

Quan es llança una excepció dins del `try`, **es comprova cada sentència *catch* en ordre i s'executa la primera el tipus de la qual coincideix amb l'excepció llançada**. Els altres blocs *catch* serán 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.

Vegem un exemple amb clàusules *catch* múltiples.

3.5 Exemple 5

Executarem diferents instruccions dins del bloc *try* i manejarem l'excepcion de divisió per zero i la de sobrepassar la grandària del vector.

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         int x, y, div, pos;
18         int[] v = {1,2,3};
19         Scanner in = new Scanner(System.in);
20
21         try
22         {
23             System.out.print("Introduce el numerador: ");
24             x = in.nextInt();
25
26             System.out.print("Introduce el denominador: ");
27             y = in.nextInt();
28
29             div = x / y;
30
31             System.out.println("La división es " + div );
32
33             System.out.print("Introduce la posición del vector a consultar: ");
34             pos = in.nextInt();
35
36             System.out.println("El elemento es " + v[pos] );
37
38         }
39         catch(ArithmeticException ex)
40         {
41             System.out.println("División por cero: " + ex);
42         }
43         catch(ArrayIndexOutOfBoundsException ex)
44         {
45             System.out.println("Sobrepasado el tamaño del vector: " + ex);
46         }
47
48         System.out.println("Fin del programa");
49     }
50 }
```

Poden succeir tres coses diferents:

- El *try* s'executa sense excepcions, s'ignoren els *catch* i s'imprimeix "Fin del programa".
- Es produeix l'excepció de divisió per zero (línia 29), el flux d'execució salta al 1er *catch*, s'imprimeix el missatge "División por cero..." i després "Fin del programa".
- Es produeix l'excepció de sobrepassar el vector (línia 36), el flux d'execució salta al 2on *catch*, s'imprimeix el missatge "Sobrepasado el tamaño del vector..." i "Fin del programa".

3.6 L'objecte Exception

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

Els dos mètodes de 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. Això és el que fa Java per defecte quan una excepció no es maneja i acaba parant el programa.

```
...
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();
}
...
```

Els objectes de tipus Exception tenen precarregat el mètode toString() pel que també és possible imprimirlos directament mitjançant println().

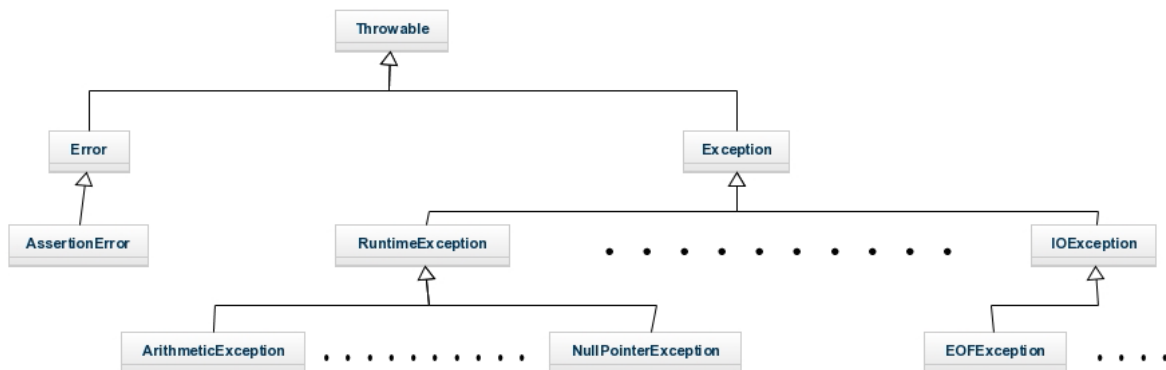
```
...
catch (Exception e){

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

}
...
```

4. JERARQUIA I TIPUS D'EXCEPCIONS JAVA

La classe `Exception` hereta de `Throwable`, i alhora, 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:

- **Excepcions comprovades:** aquelles que Java comprova durant la compilació, abans de l'execució del programa.
- **Excepcions no comprovades:** aquelles que Java no pot comprovar durant la compilació i es produiran durant l'execució del programa.

Les **excepcions comprovades** definides en *java.lang* són:

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.

Les subclasses de `RuntimeException` no comprovades són:

Excepció	Significat
<i>ArithmeticException</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 invalida.
<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>StringIndexOutOfBounds</i>	Intent de sobrepassar el límit d'una cadena.
<i>TypeNotPresentException</i>	Tipus no trobat.
<i>UnsupportedOperationException</i>	Operació no admesa.



És interessant conèixer els diferents tipus d'excepcions, però **no és necessari saber-li-les de memòria ni entendre exactament quan es produeix cadascuna.**

Encara que les excepcions que incorpora Java, gestionen la majoria dels errors més comuns, és probable que el programador preferisca crear els seus propis tipus d'excepcions per a gestionar situacions específiques de les seues aplicacions. Tan sols cal definir una subclasse de `Exception`, que naturalment és subclasse de `Throwable`.

5. DEFINIR EXCEPCIONS PRÒPIES

Quan desenvolupem programari, sobretot en desenvolupar les nostres pròpies classes, és habitual que es puguin produir excepcions que no estiguen definides dins del llenguatge Java. Per a crear una excepció pròpia hem de definir una classe derivada de la classe base *Exception*.

L'ús d'aquesta nova excepció és el mateix que hem vist.

5.1 Exemple 6

Veurem un exemple senzill de definició i ús d'un nou tipus d'excepció anomenada *ExcepcionPropia* que utilitzarem quan a se li passe a 'mètode' un valor superior a 10.

El main i el mètode que llança l'excepció:

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         try
18         {
19             metodo(1);
20             metodo(20);
21         }
22         catch (ExcepcionPropia e)
23         {
24             System.out.println("capturada :" + e);
25         }
26     }
27
28     static void metodo(int n) throws ExcepcionPropia
29     {
30         System.out.println("Llamado por metodo(" + n + ")");
31
32         if (n > 10)
33             throw new ExcepcionPropia(n);
34
35         System.out.println("Finalización normal");
36     }
37 }
```

La definició de la nova excepció.

```
12 public class ExcepcionPropia extends Exception
13 {
14     private int num;
15
16     ExcepcionPropia(int n)
17     {
18         this.num = n;
19     }
20     public String toString()
21     {
22         return "Excepcion Propia[" + this.num + "];";
23     }
24
25 }
```

Sent l'eixida:

```
run:
Llamado por metodo(1)
Finalización normal
Llamado por metodo(20)
capturada :Excepcion Propia[20]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Com es pot observar l'excepció es llança quan el número és major que 10 i serà tractada per la nova classe creada que hereta de Exception. A més se sobreescriu el mètode toString que és l'encarregat de mostrar el missatge associat a l'excepció.

6. AGRAÏMENTS

Anotacions actualitzades i adaptats al CEEDCV a partir de la següent documentació:

[1] Anotacions Programació de José Antonio Díaz-Alejo. IES Camp de Morvedre.