UD9.- Gestió d'excepcions

Mòdul: Programació
1r DAM
Curs 2018-2019

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

CONTINGUTS

- Errors i excepcions
- Excepcions en Java
- Gestió d'excepcions
  - Capturar excepcions
  - Propagar excepcions
  - Llançar excepcions
  - Crear classes d'excepcions

Errors i excepcions (I)

- **Errors**
  - Un error és un event que es produix al llarg de l'execució d'un programa i provoca una interrupció en el flux d'execució. Al produir-se esta situació, l'error genera un objecte excepció.

- **Excepcions**
  - Una excepció és un objecte generat per un error, que conté informació sobre les característiques de l'error que s'ha produït.

Errors i excepcions (II)

- Exemples d'excepcions
  - Indexació d'array fora de rang
  - Referència a cap objecte (null)
  - Errors de format
  - Errors en operacions matemàtiques (divisió per 0, ...)
  - L'arxiu que volem obrir no existix
  - Falla la connexió a una xarxa
  - La classe que es vol utilitzar no es troba en cap dels paquets utilitzats des dels import

Errors i excepcions (III)

- Fins ara els nostres programes no gestionen les excepcions, si se'n produixen:
  - 1.- S'informa per consola del problema (es mostra la traça)
  - 2.- Es para l'execució.

Errors i excepcions. Exemple

- Si executem:

```
System.out.print("Valor: ");
int valor = lector.nextInt();
System.out.print("Hemos leido : "+valor);
```

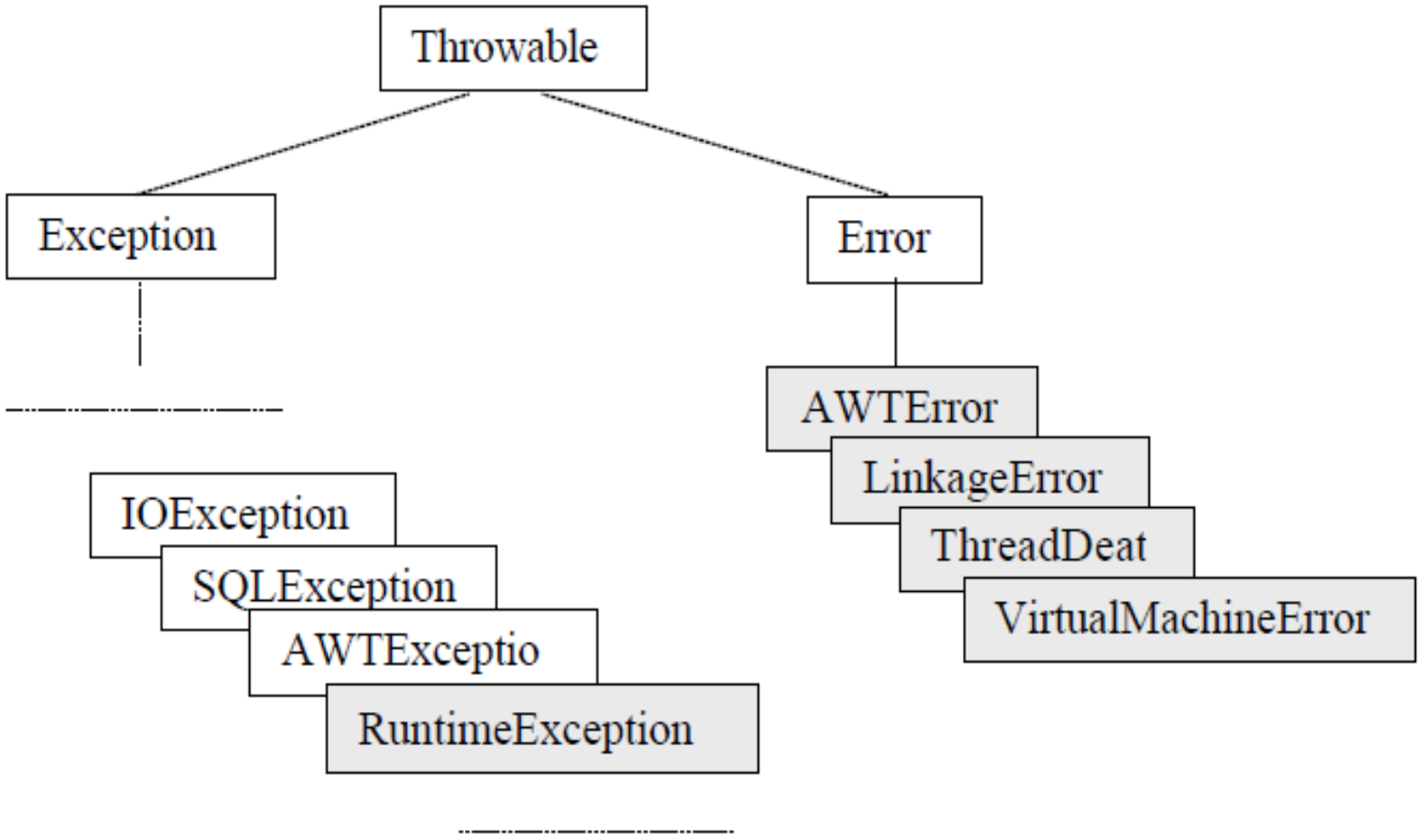
- I introduim caràcters no numèrics es llança una excepció InputMismatchException

```
Valor: hola
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:909)
        at java.util.Scanner.next(Scanner.java:1530)
        at java.util.Scanner.nextInt(Scanner.java:2160)
        at java.util.Scanner.nextInt(Scanner.java:2119)
        at unidad07.SinExcepcion1.main(SinExcepcion1.java:16)
```

Excepcions en Java (I)

- Java llança excepcions com a resposta a situacions poc habituals.
- El programador també pot llançar les seues pròpies excepcions.
- Les excepcions en Java són objectes de classes derivades de la classe base ***Exception***
- La classe ***Exception*** deriva de la classe base ***Throwable*** (java.lang.Throwable)

Excepcions en Java (II)



Excepcions en Java (III)

- Existeix tota una jerarquia de classes derivada de la classe base *Exception*.
- Les excepcions es divideixen en dos grups principals:
  - Excepcions en temps d'execució o *RuntimeExceptions* (excepcions implícites)
  - Excepcions explícites

Excepcions implícites

- Java les comprova al llarg de l'execució i les llança de forma automàtica. No cal realitzar un tractament explícit, és a dir, Java no obliga a tractar-les.
- Estes excepcions li indiquen al programador quins tipus d'errors té el programa per a que ho solucione abans de continuar.
- Exemple:
  - Quan es sobrepassa la dimensió d'un array, es llança una excepció *ArrayIndexOutOfBounds*.
  - Quan es fa ús d'una referència a un objecte que encara no ha estat creat, es llança l'excepció *NullPointerException*.

Excepcions explícites

- Java **obliga a tractar-les** si es produixen.
- Exemple:
  - *IllegalAccessException* indica que no es pot trobar un mètode.
  - IOException quan es produix error de entrada/eixida (exemple de BufferedReader).

Algunes excepcions *RuntimeException*

Classe	Situació d'excepció
NumberFormatException	Indica que una aplicació ha intentat convertir una cadena a un tipus numèric, però la cadena no té el format apropiat.
ArithmeticException	Quan ha tingut lloc una condició aritmètica excepcionl, com per exemple una divisió per zero.
ArrayStoreException	Per a indicar que s'ha intentat emmagatzemar un tipus d'objecte erroni en un array d'objectes.
IllegalArgumentException	Indica que a un mètode li han passat un argument il·legal.
IndexOutOfBoundsException	Indica que un índex d'algun tipus (un array, cadena) està fora de rang.
NegativeArraySizeException	Si una aplicació intenta crear un array amb mida negativa.
NullPointerException	Quan una aplicació intenta utilitzar *null* on es requerix un objecte.
InputMismatchException	Llançada per Scanner per a indicar que el valor recuperat no coincidix amb el patró esperat.

Excepcions i *packages*

- Les classes derivades d'Exception poden pertànyer a diferents *packages* (agrupacions de classes) de Java, depenent del tipus d'excepció que representen:
  - package java.lang
  - package java.io
  - package java.util
  - ...

Capturar excepcions

- Detecta i respon a errors mentre s'executa una aplicació.
- Utilitza ***try***...***catch***...***finally*** per encapsular i protegir blocs de codi que podrien provocar errors:
  - Cada bloc té un o més controladors associats.
  - Cada controlador especifica alguna forma de condició de filtre en el tipus d'excepció que controla.
- Avantatges:
  - Permet la separació entre la lògica i el codi de gestió d'errors.
  - Facilita la lectura, depuració i manteniment del codi.

Com utilitzar l'estructura *try...catch* (I)

- Posar el codi que podria llançar excepcions en un bloc try.
- Gestionar les excepcions en un o més blocs catch.

```
try {
   codi que pot provocar errors

} catch (ExceptionA a) {
  ....
} catch (ExceptionB b) {
  ....
}
```

Com utilitzar l'estructura *try...catch* (II)

- Des del bloc *catch* es maneja l'excepció
- Cada *catch* maneja un tipus d'excepció.
- Quan es produix una excepció, es busca el primer *catch* que utilitze el mateix tipus d'excepció que s'ha produït:
  - L'últim *catch* ha de ser el que capture excepcions genèriques i els primers han de ser els més específics.
  - Si anem a tractar totes les excepcions (siguen del tipus que siguen), hem de pensar si potser amb un catch que capture objectes *Exception* n'hi ha prou.

Com utilitzar l'estructura *try...catch* (III)

```
String[] texto = {"Uno", "Dos", "Tres", "Cuatro", "Cinco"};

for (int i = 0; i < 10; i++) {

   try {

      System.out.println("indice " + i + " = " + texto[i]);

   } catch (ArrayIndexOutOfBoundsException e) {

      System.out.println("Fallo en el índice " + i);
   }
}
```

Com utilitzar l'estructura *try...catch* (IV)

```
try {
   System.out.print("Valor: ");
   int valor = lector.nextInt();
   int auxiliar = 8 / valor;
   System.out.println(auxiliar);

} catch (ArithmeticException e) {
   if(valor == 0)
      System.out.println("Division por cero");
   else
      System.out.println("Excepción aritmética");
}
```

Com utilitzar l'estructura *try...catch* (V)

```
try {
    System.out.print("Valor: ");
    int valor = lector.nextInt();
    int auxiliar = 8 / valor;
    System.out.println(auxiliar);
} catch (ArithmeticException e1) {
    System.out.println("Division por cero");
} catch (InputMismatchException e2) {
    System.out.println("No se ha leído un entero....");
} catch (Exception e9) {
    System.out.println("Error general");
} finally {
    lector.nextLine();
}
```

Com utilitzar l'estructura *try...catch* (VI)

```
int valor = 0;
boolean leido = false;
Scanner lector = new Scanner(System.in);
do {
    try {
        System.out.print("Entra un número entero: ");
        valor = lector.nextInt();
        leido = true;
    } catch (InputMismatchException e) {
        System.out.println("Error en la introducción del número");
        lector.nextLine(); //vaciamos el buffer de entrada
    }
} while (!leido);
System.out.println("Hemos leido : " + valor);
```

Com utilitzar el bloc *finally*

- És opcional; si s'inclou, s'executa sempre.
- Emprar-lo, per exemple, per colocar codi de “neteja”, com el que s'empra per tancar arxius.

```
try {
  //codi que pot provocar errors

} catch (ExceptionA a) {
 ....

} catch (ExceptionB b) {
 ....

} finally {

  //codi que s'executa sempre
}
```

Alguns mètodes d'*Exception*

- **String getMessage()**
  - Recupera el missatge descriptiu de l'excepció o una indicació específica de l'error produït.

```
try{
....
} catch (IOException ioe){
    System.out.println(ioe.getMessage());
}
```

- **String toString()**
  - Converteix a cadena informació l'error. Normalment conté la classe d'excepció i el text de getMessage().

```
try{
....
} catch (IOException ioe){
    System.out.println(ioe.toString());
}
```

- **void printStackTrace()**
  - Escriu la traça d'invocacions que ha provocat l'excepció i el seu missatge associat (és el que s'anomena informació de pila).
  - És el mateix missatge que mostra l'executor (màquina virtual de Java) quan no es controla l'excepció.

Directrius per a la captura d'excepcions

- No emprar la gestió estructurada d'excepcions per a errors que es produïsquen de forma rutinària. En estos casos és millor utilitzar altres blocs de codi per controlar estos errors.
  - if ...else if, etc.
- Organitzar els blocs *catch* des dels més específics fins als més generals.

Propagar excepcions

- Quan es produix un error:
  - Es pot capturar en el mètode en el qual s'ha produït o
  - es pot propagar cap al mètode invocador per a que el capture. Si este no el captura, es propaga successivament fins al main() i el main(), si no el captura, el propaga fins al sistema operatiu.
- Sols es propaguen els errors que deriven de la classe RuntimeException. La resta no es prograpaguen i s'han de capturar o llançar.

Exemple (I)

```
import java.util.Scanner;

public class Prova1 {

  public static int llegirNumero(){

    Scanner reader=new Scanner(System.in);

    int num=0;

    try {

      System.out.print("Inserta un número: ");

      num=Integer.parseInt(reader.nextLine());

      System.out.println("Número correcte");

    } catch (NumberFormatException e){

      System.out.println("Sols es poden introduir números.");

    }

   return num;

  }

  public static void main(String[] args) {

    try {

      llegirNumero();

    } catch (NumberFormatException e){

      System.out.println("Captura de l'excepció des del main.");

    }

  }

} // Fi de la classe Prova1
```

Què mostra este programa si introduim una lletra?

Exemple (II)

```
import java.util.Scanner;

public class Prova1 {

  public static int llegirNumero(){

    Scanner reader=new Scanner(System.in);

    int num=0;

    System.out.print("Inserta un número: ");

    num=Integer.parseInt(reader.nextLine());

    System.out.println("Número correcte");

    return num;

  }

  public static void main(String[] args){

    try{

      llegirNumero();

    } catch (NumberFormatException e){

      System.out.println("Captura de l'excepcio des
del main.");

    }

  }

} // Fi de la classe Prova1
```

Què mostra ara el programa si introduim una lletra?

Llançar excepcions (I)

- Situació 1.- Dins d'un mètode on es pot produir un dels errors que Java obliga a controlar (per exemple, *IOException*). Dos opcions:
  - Capturar i tractar l'error amb *try-catch*
  - Llançar l'error. S'expressa de la següent forma en la capçalera del mètode:

```
tipoRetornat nomMètode (paràmetres) throws Excepció
tipoRetornat nomMètode (paràmetres) throws Excepció1,Excepció2...
```

- Els errors que deriven de la classe **RuntimeException** (per exemple, NumberFormatException) no cal llançar-los, ja que en cas de que es produïsquen, **es propaguen automàticament** si no han estat capturats.

Llançar excepcions (II)

- Situació 2.- Dins d'un mètode volem controlar un error, però també vullguem controlar-lo fora d'eixe mètode. Per a fer-ho:
  - Capturar l'error dins del mètode (amb el **try-catch**).
  - Dins del catch, llançar l'error per a que el puga rebre el mètode invocador.
  - Un error es llança de la següent forma:

```
throw (nomObjecteTipusExcepció);
```

  - La instrucció throw provoca que s'abandone l'execució del mètode on es troba i passe el control al mètode invocador, al catch que captura l'error.

Crear classes d'excepcions

- Ha de ser una subclasse de *java.lang.Throwable*
  - Exemple: `class MyMistakes extends Exception {...}`

- Acostuma a incloure informació de per què es crea (circunstància excepcional que provoca excepció)

```
class MyMistakes extends Exception {

      public MyMistakes(String msg){
         super(msg);
      }
}
```