

Programació



UT10. Control d'excepcions

Introducció

- Els errors en temps d'execució ocorren quan la JVM detecta una operació que és impossible realizar (per exemple, ***ArrayIndexOutOfBoundsException*** o ***NullPointerException***)
- En el desenvolupament de programes, en qualsevol llenguatge de programació el programador ha de disposar de mecanismes (proporcionats pel llenguatge) per a detectar errors que es puguen produir **en temps d'execució**.

Gestió d'excepcions

- En Java, els errors en temps d'execució són tractats com a excepcions.
- Una excepció és un objecte que representa un error o una condició que **prevé l'execució** per a procedir amb normalitat.
- Anomenem gestió (o control) d'excepcions al **conjunt de mecanismos que un llenguatge de programación proporciona per a detectar i gestionar els errors que es puguen producir en temps d'execució**.

```
12.5
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:840)
at java.util.Scanner.next(Scanner.java:1461)
at java.util.Scanner.nextFloat(Scanner.java:2319)
at Comprobante.main(Comprobante.java:46)
```

Vista prèvia del control de excepcions

- Per a veure més clar com es realitza el tractament d'excepcions, incloent com un objecte excepció es crea i es llança, anem a veure el següent exemple:

```
import java.util.Scanner;

public class Quotient {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        System.out.println(number1 + " / " + number2 + " is " +
            (number1 / number2));
    }
}
```

Vista prèvia del control de excepcions

- Si introduceixen un 5 i un 2, el resultat és 2
- Si introduceixen un 3 i un 0 el resultat és:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at Quotient.main(Quotient.java:11)
```

- En eixe cas, la **JVM** ha sigut la que ha llançat l'excepció i el programador no té control sobre eixe error.

Vista prèvia del control de excepcions

- Una manera d'evitar l'error seria fer un simple if que controlara eixa situació.

```
import java.util.Scanner;

public class QuotientWithIf {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        if (number2 != 0)
            System.out.println(number1 + " / " + number2
                + " is " + (number1 / number2));
        else
            System.out.println("Divisor cannot be zero ");
    }
}
```

Sense maneig d'excepcions

- Sense utilitzar excepcions, podríem arribar també a la següent solució.
- Però d'esta manera, estaria també acabant el programa.

```
import java.util.Scanner;

public class QuotientWithMethod {
    public static int quotient(int number1, int number2) {
        if (number2 == 0) {
            System.out.println("Divisor cannot be zero");
            System.exit(1);
        }

        return number1 / number2;
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        int result = quotient(number1, number2);
        System.out.println(number1 + " / " + number2 + " is "
            + result);
    }
}
```

Captura i tractament

- Partim de la terminologia en Java de que generar una excepció és "**llançar una excepció**".
- En Java disposem del mecanisme "try-catch" per a capturar una excepció llançada i determinar l'actuació següent que corresponga.



Control d'excepcions

- Java disposa d'un **procés per a llançar una excepció** que pot ser capturada i gestionada de la manera més convenient per el programador.

```
import java.util.Scanner;

public class QuotientWithException {
    public static int quotient(int number1, int number2) {
        if (number2 == 0)
            throw new ArithmeticException("Divisor cannot be zero");

        return number1 / number2;
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter two integers
        System.out.print("Enter two integers: ");
        int number1 = input.nextInt();
        int number2 = input.nextInt();

        try {
            int result = quotient(number1, number2);
            System.out.println(number1 + " / " + number2 + " is "
                + result);
        }
        catch (ArithmeticException ex) {
            System.out.println("Exception: an integer " +
                "cannot be divided by zero ");
        }

        System.out.println("Execution continues ...");
    }
}
```

Tipus d'excepcions

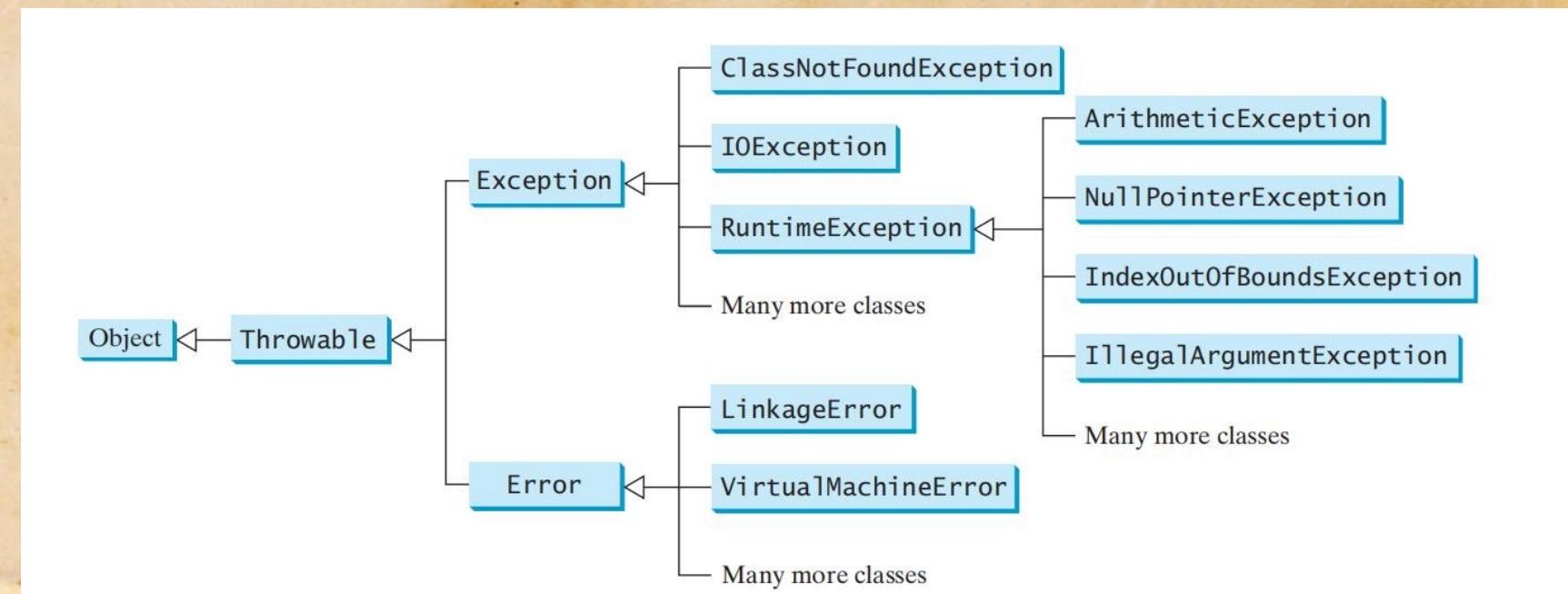
- En Java es distingeixen entre errors i excepcions.
- **Errors:** corresponen a **situacions irrecuperables**, que no tenen solució i que no depenen del programador, el qual no s'ha de preocupar de tractar (per exemple: falta de recursos, canal de E/S no respon, etc.)
- **Excepcions:** **situacions excepcionals** que els programes poden trobar en temps d'execució (normalment per culpa del programador).

Classe Error i classe Exception

- Java engloba tots els possibles errors en classes derivades de la classe Error i totes les possibles excepcions en classes derivades de la classe Exception.
- És a dir, quan es produeix un error, es crea un objecte de la subclass de Error o Exception corresponent que conté la informació del context en que s'ha produït el problema.

Classe Throwable

- Les classes Error i Exception deriven ambdues de la classe Throwable (llançable) la qual proporciona mecanismes comuns per la gestió de qualsevol tipus d'error i excepció:
 - Constructors
 - Mètodes comuns



Quines excepcions es llancen?

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(1 / 0);  
    }  
}
```

(a)

```
public class Test {  
    public static void main(String[] args) {  
        int[] list = new int[5];  
        System.out.println(list[5]);  
    }  
}
```

(b)

```
public class Test {  
    public static void main(String[] args) {  
        String s = "abc";  
        System.out.println(s.charAt(3));  
    }  
}
```

(c)

```
public class Test {  
    public static void main(String[] args) {  
        Object o = new Object();  
        String d = (String)o;  
    }  
}
```

(d)

```
public class Test {  
    public static void main(String[] args) {  
        Object o = null;  
        System.out.println(o.toString());  
    }  
}
```

(e)

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(1.0 / 0);  
    }  
}
```

(f)

Constructors de Throwable

- **Throwable()**
 - Construeix un objecte amb el missatge a null
- **Throwable(String message)**
 - Construeix un objecte, sent el seu missatge l'indicat
- **Throwable(String message, Throwable causa)**
 - Construeix un objete, sent el seu missatge l'indicat i la causa que ha provocat l'excepció (un altre objecte que ha provocat que ocórrega este)
- **Throwable(Throwable causa)**
 - Construeix un objecte amb la causa que ha provocat l'excepció i com a missatge, el missatge que tinga l'objecte “causa”.

Constructors de Throwable

- Fixa't que dos dels constructors indicats anteriorment reben un altre objecte Throwable com causant del nou objecte.
- Això permet realitzar un **encadenament d'excepcions**.

Mètodes comuns (mètodes de Throwable)

public Throwable getCause()

- Retorna la causa de l'excepció o null.

public String getMessage()

- Retorna el missatge que defineix l'excepció o null.

public void printStackTrace()

- Visualitza per consola, el context on s'ha produït l'error i la cascada de crides des del mètode main que porta al punt on s'ha produït l'error.

public String toString()

- Retorna una descripción curta de l'objecte.

Classe Exception

- Per a realitzar una bona gestió d'excepcions, en primer lloc ens hem de centrar en la jerarquía de classes que naixen a partir d'Exception.



Subtipus de Exceptió

- Cap diferenciar dos grans subtipus de Exception:
 - **Excepcions implícites que el programador NO té obligació de capturar i gestionar.** Estan agrupades en la classe RuntimeException i normalment estan relacionats amb errors de programació:
 - **Errors que normalment no es revisen** al codi de producció d'un programa (per exemple, revisar que una referència no és nul·la).
 - **Errors que el programador hauria d'haver revisat al escriure el codi de producció** (per exemple, accedir a una posició d'un array inexistent).

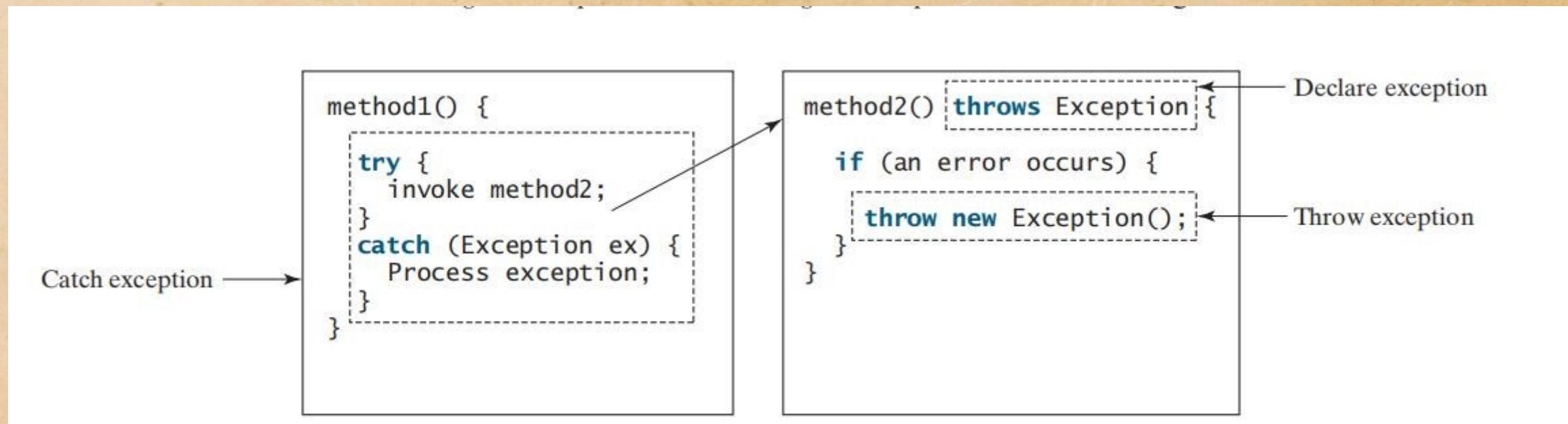
Subtipus d'Exception

- **Excepcions explícites** (totes les de la classe Exception que no pertanyen a la classe RuntimeException) **que el programador està obligat a tenir en compte** allà on es puguen produir.



Captura i tractament

- La sintaxi general és la següent:



Declaració d'excepcions en mètodes

- Cada mètode podrà **llançar** una sèrie d'excepcions i s'hauran de definir aquelles que pot llançar.
- Tal i com hem comentat abans, **Java no ens obliga** a gestionar els Error o els RuntimeException expressament.
- El resta de excepcions deuen ser declarats al mètode com excepcions a gestionar (throws).

Declaració d'excepcions

- Per a declarar que un mètode llançarà dins del seu codi una excepció, s'utilitza la paraula reservada throws

```
public void myMethod()  
    throws Exception1, Exception2, ..., ExceptionN
```

- Per exemple

```
public void myMethod() throws IOException
```

Llançant excepcions

- Un programa que detecta un error pot crear una instància d'una excepció del tipus apropiat. (Automàticament, sense fer ús de “throw”)
- També es poden llançar manualment. Per exemple, un mètode que requereix treballar amb només números positius rep un número negatiu. Es podria decidir crear una excepció IllegalArgumentException:

```
throw new IllegalArgumentException("Wrong Argument");
```

No confondre `throw` amb `throws`

Capturant excepcions

- Ja sabem com llançar una excepció. Ara només queda recollir-la i tractar-la de la manera convenient. Per això disposem de la sentència try-catch

```
try {  
    statements; // Statements that may throw exceptions  
}  
catch (Exception1 exVar1) {  
    handler for exception1;  
}  
catch (Exception2 exVar2) {  
    handler for exception2;  
}  
...  
catch (ExceptionN exVarN) {  
    handler for exceptionN;  
}
```

Captura i tractament (A tindre en compte)

- Observa que es poden definir diversos blocs "catch".
- En cas de produir-se una excepció, **s'avorta l'execució del bloc try i es comencen a avaluar els diferents blocs catch en l'ordre que estiguem definits**, executant-se aquell que pertanya a la classe o classe superior de l'excepció llançada. (per això és important declarar primer "catch" amb classes més concretes i després més generals)

Exemple

```
import java.io.*;

public class NewClass {

    public static void main(String[] args) {

        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Introdueix el total de paràmetres que vols mostrar");

        try {
            int total = Integer.parseInt(bf.readLine());
            for (int i = 0; i < total; i++) {
                System.out.println(args[i]);
            }
        } catch (IOException e) {
            System.out.println("Error d'entrada de dades");
        } catch (NumberFormatException e) {
            System.out.println("La dada no té format numèric vàlid");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("No s'han introduït tants arguments al iniciar el
programa");
        }
    }
}
```

Captura i tractament

- En cas d'existir un bloc "catch" que corresponga amb l'excepció produïda en el bloc "try", s'executarà eixe bloc "catch" (que pot estar buit).

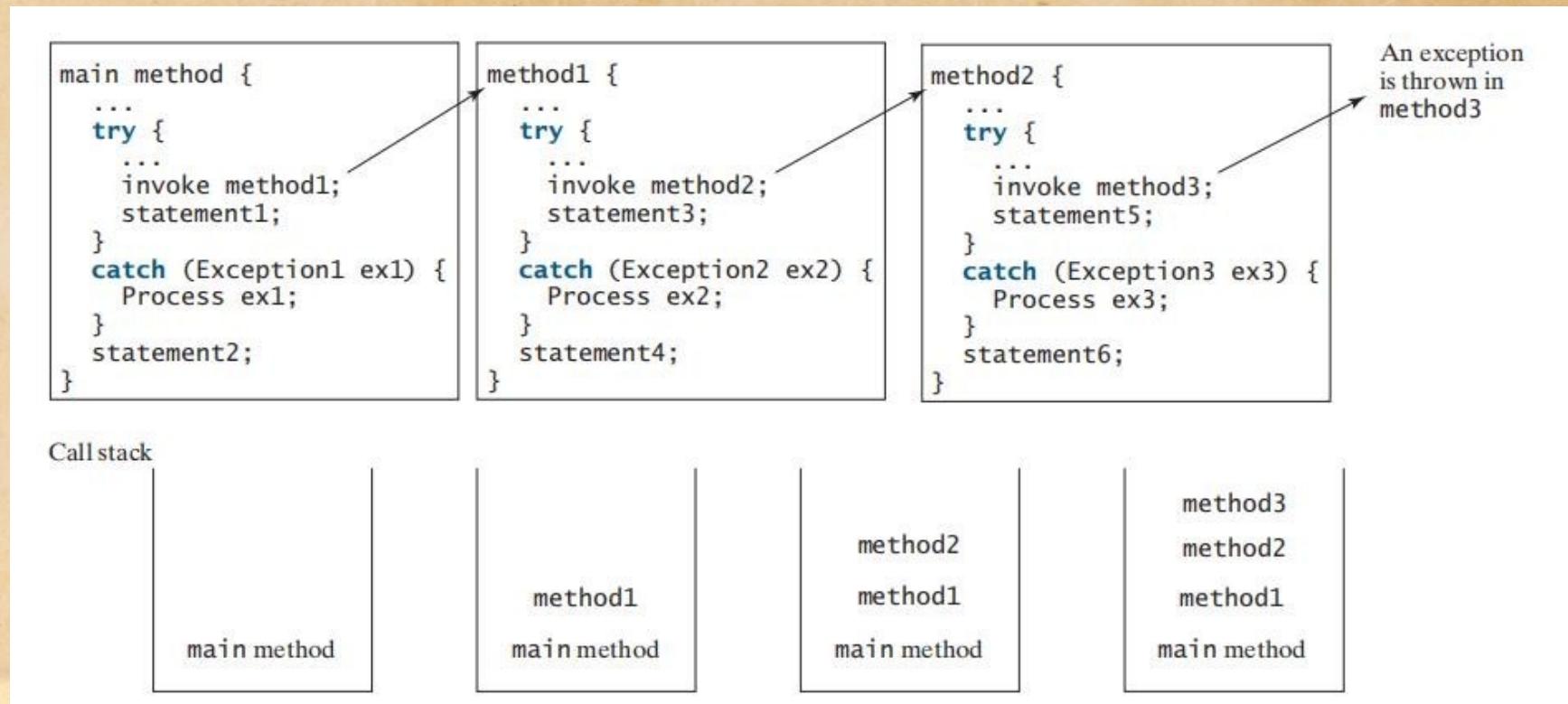
Captura i tractament

- En cas d'ocórrer l'excepció i no existir cap bloc "catch" que puga ser executat, l'excepció es propaga al mètode que va invocar al que ha produït l'excepció que igualment podrà capturar l'excepció o no (propagar-la).
- Si l'excepció es propaga fins el mètode main() sense ser tractada, es produirà una finalització anormal del programa.

```
12.5
Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:840)
  at java.util.Scanner.next(Scanner.java:1461)
  at java.util.Scanner.parseFloat(Scanner.java:2319)
  at Comprobante.main(Comprobante.java:46)
```

Què passa si...

- L'excepció és del tipus Exception3? la excepció és del tipus Exception2? la excepció és del tipus Exception1?



Recomanació d'ordre "catch"

Es recomana ordenar els blocs "catch" demajor a menor especificitat. (més concreta a més general)

```
try {  
    ...  
}  
catch (Exception ex) {  
    ...  
}  
catch (RuntimeException ex) {  
    ...  
}
```

(a) Wrong order

```
try {  
    ...  
}  
catch (RuntimeException ex) {  
    ...  
}  
catch (Exception ex) {  
    ...  
}
```

(b) Correct order

Preguntes

- Suposem que statement2 provoca una excepció:
 - S'executa statement3?
 - Si l'excepció no es captura, s'executa statement4?
 - Si l'excepció és capturada, s'executa statement4?

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex1) {  
}  
catch (Exception2 ex2) {  
}  
  
statement4;
```

Més preguntes

Què es mostra per pantalla?

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            int[] list = new int[10];  
            System.out.println("list[10] is " + list[10]);  
        }  
        catch (ArithmaticException ex) {  
            System.out.println("ArithmaticException");  
        }  
        catch (RuntimeException ex) {  
            System.out.println("RuntimeException");  
        }  
        catch (Exception ex) {  
            System.out.println("Exception");  
        }  
    }  
}
```

Més preguntes

Què es mostra per pantalla?

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            method();  
            System.out.println("After the method call");  
        }  
        catch (ArithmetricException ex) {  
            System.out.println("ArithmetricException");  
        }  
        catch (RuntimeException ex) {  
            System.out.println("RuntimeException");  
        }  
        catch (Exception e) {  
            System.out.println("Exception");  
        }  
    }  
  
    static void method() throws Exception {  
        System.out.println(1 / 0);  
    }  
}
```

Més preguntes

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            method();  
            System.out.println("After the method call");  
        }  
        catch (RuntimeException ex) {  
            System.out.println("RuntimeException in main");  
        }  
        catch (Exception ex) {  
            System.out.println("Exception in main");  
        }  
    }  
  
    static void method() throws Exception {  
        try {  
            String s = "abc";  
            System.out.println(s.charAt(3));  
        }  
        catch (RuntimeException ex) {  
            System.out.println("RuntimeException in method()");  
        }  
        catch (Exception ex) {  
            System.out.println("Exception in method()");  
        }  
    }  
}
```

I aquí què es mostra per pantalla?

Captura i tractament - bloc finally

- **El bloc "finally" (si s'ha definit) s'executa sempre, independentment de produir-se l'excepció o no.**
- S'executa inclusivament si dins del bloc "try- catch" hi ha alguna setència del tipus break o return.
- La única situació en que NO s'executarà el bloc finally és quan executem el mètode System.exit().

Captura i tractament

- En la majoria de casos, un bloc try sempre anirà acompanyat de un bloc catch.
- Pot passar que no hi haja cap catch definit, només per assegurar l'execució de certes accions, a través d'un bloc "finally".
- No és possible definir un bloc "try" sense "catch" ni "finally"

Captura i tractament

Exemple: try sense catch

```
try {  
    obrirAixeta();  
    regarGespa();  
}  
  
finally {  
    tancarAixeta();  
}
```

Què passa si...

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex1) {  
}  
finally {  
    statement4;  
}  
statement5;
```

- Si no passa cap excepció, s'executarà statement4 i statement5?
- Si l'excepció és Exception1, s'executarà statement4 i statement5?
- Si no és del tipus Exception1, s'executarà statement4 i statement5?

Rellançar excepcions

- La manera de rellançar una excepció seria:

```
try {  
    statements;  
}  
catch (TheException ex) {  
    perform operations before exits;  
    throw ex;  
}
```

- Al rellançar l'excepció, el programa buscarà un nou catch que reculla l'excepció llançada (l'actual ja no s'inclou com a possible catch)

S'executarà statement4 i statement5 si....

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex1) {  
}  
catch (Exception2 ex2) {  
    throw ex2;  
}  
finally {  
    statement4;  
}  
statement5;
```

- no passa una excepció?
- si l'excepció és del tipus Exception1?
- si la excepció és del tipus Exception2?
- si no és ni Exception1 ni Exception2?

Encadenar excepcions

- Hem vist com rellançar una excepció, però també és possible llançar una excepció nova dins d'un catch. Ho anomenem excepcions encadenades.

```
java.lang.Exception: New info from method1
at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:16)
at ChainedExceptionDemo.main(ChainedExceptionDemo.java:4)
Caused by: java.lang.Exception: New info from method2
at ChainedExceptionDemo.method2(ChainedExceptionDemo.java:21)
at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:13)
... 1 more
```

```
public class ChainedExceptionDemo {
    public static void main(String[] args) {
        try {
            method1();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static void method1() throws Exception {
        try {
            method2();
        }
        catch (Exception ex) {
            throw new Exception("New info from method1", ex);
        }
    }

    public static void method2() throws Exception {
        throw new Exception("New info from method2");
    }
}
```

Exemple 1. Excepció no capturada

```
//Fitxer Excepcio01.java

public class Excepcio01 {
    public static void main(String args[]) {
        String t[]={ "Hola", "Adéu", "Fins demà" };
        for (int i=0; i<=t.length; i++)
            System.out.println("Posició " + i + " : " + t[i]);
        System.out.println("El programa s'ha acabat.");
    }
}
```

Exemple 2. Excepció capturada

```
//Fitxer Excepicio02.java

public class Excepicio02 {
    public static void main(String args[]) {
        String t[]={ "Hola", "Adéu", "Fins demà" };
        try {
            System.out.println("Abans d'executar el for");
            for (int i=0; i<=t.length; i++)
                System.out.println("Posició " + i + " : " + t[i]);
            System.out.println("Després d'executar el for");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("El programador estava a la lluna... S'ha sortit de
                            límits!!!");
        }
        System.out.println("Final del programa");
    }
}
```

Ejemplo 3. Captura incorrecta

```
//Fitxer Excepcio03.java

public class Excepcio03 {
    public static void main(String args[]) {
        String t[]={ "Hola", "Adéu", "Fins demà" };
        try {
            System.out.println("Abans d'executar el for");
            for (int i=0; i<=t.length; i++)
                System.out.println("Posició " + i + " : " + t[i]);
            System.out.println("Després d'executar el for");
        } catch (StringIndexOutOfBoundsException e) {
            System.out.println("El programador estava a la lluna... S'ha sortir de
                límits!!!");
        } finally {
            System.out.println("Aquest codi s'executa, peti qui peti!!!!");
        }
        System.out.println("Final del programa");
    }
}
```

Exemple 4. Excepció sense capturar en mètode interior

```
//Fitxer Excepcio04.java

public class Excepcio04 {
    public static void met02() {
        String t[]={ "Hola", "Adéu", "Fins demà"};
        for (int i=0; i<=t.length; i++)
            System.out.println("Posició " + i + " : " + t[i]);
        System.out.println("El mètode met02 s'ha acabat.");
    }

    public static void met01() {
        System.out.println("Entrem en el mètode met01 i anem a executar met02");
        met02();
        System.out.println("Tornem a estar en met02 després de finalitzar met02")
        ;
    }

    public static void main(String args[]) {
        System.out.println("Iniciem el programa i anem a executar met01");
        met01();
        System.out.println("Tornem a estar en el main després de finalitzar met01
        ");
    }
}
```

Exemple 5. Excepció capturada en mètode

```
//Fitxer Excepcio05.java

public class Excepcio05 {
    public static void met03() {
        String t[]={ "Hola", "Adéu", "Fins demà"};
        for (int i=0; i<t.length; i++)
            System.out.println("Posició " + i + " : " + t[i]);
        System.out.println("El mètode met03 s'ha acabat.");
    }

    public static void met02() {
        System.out.println("Entrem en el mètode met02 i anem a executar met03");
        met03();
        System.out.println("Tornem a estar en met02 després de finalitzar met03")
        ;
    }

    public static void met01() {
        try {
            System.out.println("Entrem en el mètode met01 i anem a executar met02
                ");
            met02();
            System.out.println("Tornem a estar en met01 després de finalitzar
                met02");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("El programador estava a la lluna... S'ha sortir de
                límits!!!");
        }
    }

    public static void main(String args[]) {
        System.out.println("Iniciem el programa i anem a executar met01");
        met01();
        System.out.println("Tornem a estar en el main després de finalitzar met01
            ");
    }
}
```

Exemple. Obtenció d'informació d'una excepció

```
static void met01()
{
    try
    {
        System.out.println("Entrem en el mètode met01 i anem a executar met02");
        met02();
        System.out.println("Tornem a estar en met01 després de finalitzar met02")
        ;
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
        System.out.println("Estem dins el bloc catch que ha capturat l'excepció
        .");
        System.out.println("Informació que dona el mètode getMessage():");
        System.out.println(e.getMessage());
        System.out.println("Informació que dona el mètode printStackTrace():");
        e.printStackTrace();
        System.out.println("Informació que dona el mètode toString():");
        System.out.println(e);
    }
}
```

Gestió: ¿captura o delegació?

- Java obliga al programador a gestionar totes les excepcions excepte les derivades de `RuntimeException`.
- Per tant, el programador està obligat a saber quines excepcions ha que controlar, donat un mètode determinat.

Maneres de saber si un mètode gestiona excepcions

- Primera: Fixant-nos en la documentació del mètode (API)

Scanner

```
public Scanner(Path source)
    throws IOException
```

Constructs a new Scanner that produces values scanned from the specified file. Bytes from the file are converted into characters using the underlying platform's default charset.

Parameters:

source - the path to the file to be scanned

Throws:

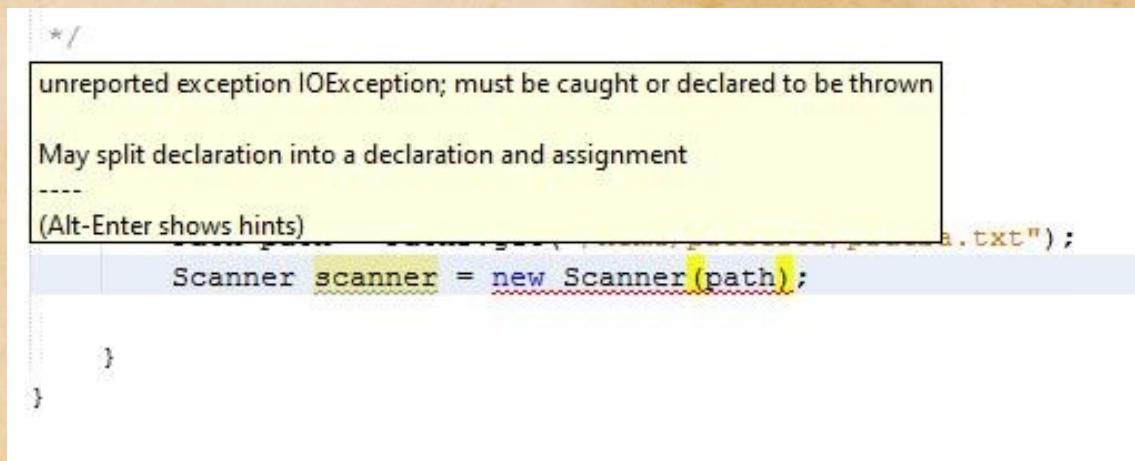
IOException - if an I/O error occurs opening source

Since:

1.7

Maneres de saber si un mètode gestiona excepcions

- Segona: observant els errors de compilació que apareixen si no es gestiona una excepció de gestió obligatòria (les que no deriven de RuntimeException).



The screenshot shows a Java code editor with the following code:

```
/*
 * 
 */
unreported exception IOException; must be caught or declared to be thrown
May split declaration into a declaration and assignment
-----
(Alt-Enter shows hints)
    Scanner scanner = new Scanner(path);
}
}
```

A tooltip is displayed over the line `Scanner scanner = new Scanner(path);`, containing the following text:

- unreported exception IOException; must be caught or declared to be thrown
- May split declaration into a declaration and assignment
-
- (Alt-Enter shows hints)

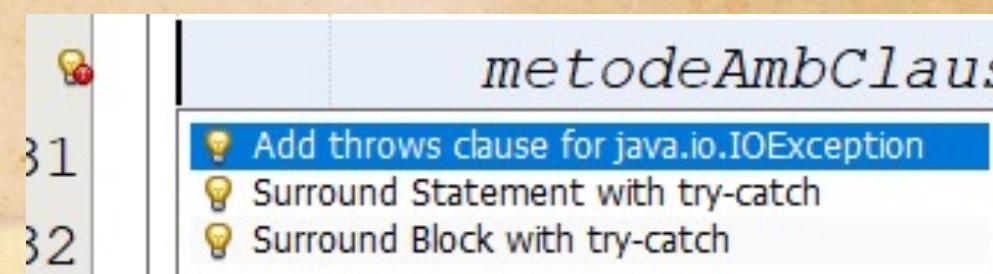
Exemple de delegació de gestió d'excepció al mètode superior

```
public static void metodeAmbClausulaThrows (String rutaAFitxer) throws IOException{  
    Path path = Paths.get(rutaAFitxer);  
    Scanner scanner = new Scanner(path);  
    scanner.close();  
}
```

```
public static void main(String[] args) {  
    metodeAmbClausulaThrows("/home/ciclost/prova.txt");  
}
```

unreported exception IOException; must be caught or declared to be thrown

(Alt-Enter shows hints)



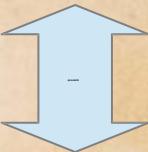
Llançament d'excepcions en mètodes propis

- Els mètodes de l'API de Java llancen excepcions segons les seues necessitats.
- De la mateixa manera, els nostres mètodes també poden llançar-los.

Realitzar el llançament

- Primer: Es crea un objecte de l'excepció que es vulga llançar.
- Segon: Es llança l'excepció fent ús de la clàusula throw seguida de l'objecte a llançar.

```
nomExcepció e = new nomExcepció (...);  
throw e;
```



```
throw new nomExcepció (...);
```

Exemple. Llançament d'una excepció de gestió obligatòria

```
//Fitxer Excepcio09.java

public class Excepcio09 {

    /* Mètode que avalua si la taula t té n cel·les, provocant, en cas de ser
       evaluada com a fals,
       una excepció d'obligada gestió: Exception */
    public static void verificaLengthTaula (int n, String t[]) throws Exception
    {
        if (t.length!=n) throw new Exception ("La taula no té la llargada
            indicada.");
        System.out.println ("Sortida de verificaLengthTaula.");
    }

    public static void main (String args[]) {
        try {
            System.out.println("Punt 1.");
            verificaLengthTaula (4, new String[4]);
            System.out.println("Punt 2.");
            verificaLengthTaula (2, new String[4]);
            System.out.println("Punt 3.");
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println ("Programa finalitzat.");
    }
}
```

Gestió d'excepcions llançades



- En el moment que es llança una excepció es deu optar per:

1) Gestionar la excepció dins del mateix mètode

2) Delegar la gestió de la excepció a mètodes superiors.

És millor gestionar l'error al mètode principal? Al mètode concret que provoca l'error? Un intermedi?...

Exemple llançament excepció de gestió no obligatòria

```
//Fitxer Excepcio10.java

public class Excepcio10 {

    /* Mètode que avalua si la taula t té n cel·les, provocant, en cas de ser
       evaluada com a fals,
       una excepció RuntimeException */
    public static void verificaLengthTaula (int n, String t[]) {
        if (t.length!=n) throw new RuntimeException ("La taula no té la llargada
            indicada.");
        System.out.println ("Sortida de verificaLengthTaula.");
    }

    public static void main (String args[]) {
        System.out.println("Punt 1.");
        verificaLengthTaula (4, new String[4]);
        System.out.println("Punt 2.");
        verificaLengthTaula (2, new String[4]);
        System.out.println("Punt 3.");
        System.out.println ("Programa finalitzat.");
    }
}
```

Definir excepcions pròpies

- Java defineix un número limitat d'excepcions, però pot passar que segons el context necessitem crear la nostra pròpia situació excepcional que l'API de Java no considera com tal.
- Per això es poden definir excepcions pròpies
- Es crearà una classe que deriva d'Exception o de qualsevol de les seues classes derivades, si és necessari (més comú és la primera opció).

Ejemplo

```
public class InvalidRadiusException extends Exception {  
    private double radius;  
  
    /** Construct an exception */  
    public InvalidRadiusException(double radius) {  
        super("Invalid radius " + radius);  
        this.radius = radius;  
    }  
  
    /** Return the radius */  
    public double getRadius() {  
        return radius;  
    }  
}
```

Constructors
més habituals

java.lang.Exception

+Exception()
+Exception(message: String)

Suposem que el mètode getRadius llança una excepció InvalidRadiusException. Què mostrarà el programa per pantalla?

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            method();  
            System.out.println("After the method call");  
        }  
        catch (RuntimeException ex) {  
            System.out.println("RuntimeException in main");  
        }  
  
        catch (Exception ex) {  
            System.out.println("Exception in main");  
        }  
    }  
}
```

```
static void method() throws Exception {  
    try {  
        Circle c1 = new Circle(1);  
        c1.setRadius(-1);  
        System.out.println(c1.getRadius());  
    }  
    catch (RuntimeException ex) {  
        System.out.println("RuntimeException in method()");  
    }  
    catch (Exception ex) {  
        System.out.println("Exception in method()");  
        throw ex;  
    }  
}
```

Tots els constructors d'Exception



Constructor and Description

`Exception()`

Constructs a new exception with `null` as its detail message.

`Exception(String message)`

Constructs a new exception with the specified detail message.

`Exception(String message, Throwable cause)`

Constructs a new exception with the specified detail message and cause.

`Exception(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)`

Constructs a new exception with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled.

`Exception(Throwable cause)`

Constructs a new exception with the specified cause and a detail message of (`cause==null ? null : cause.toString()`) (which typically co

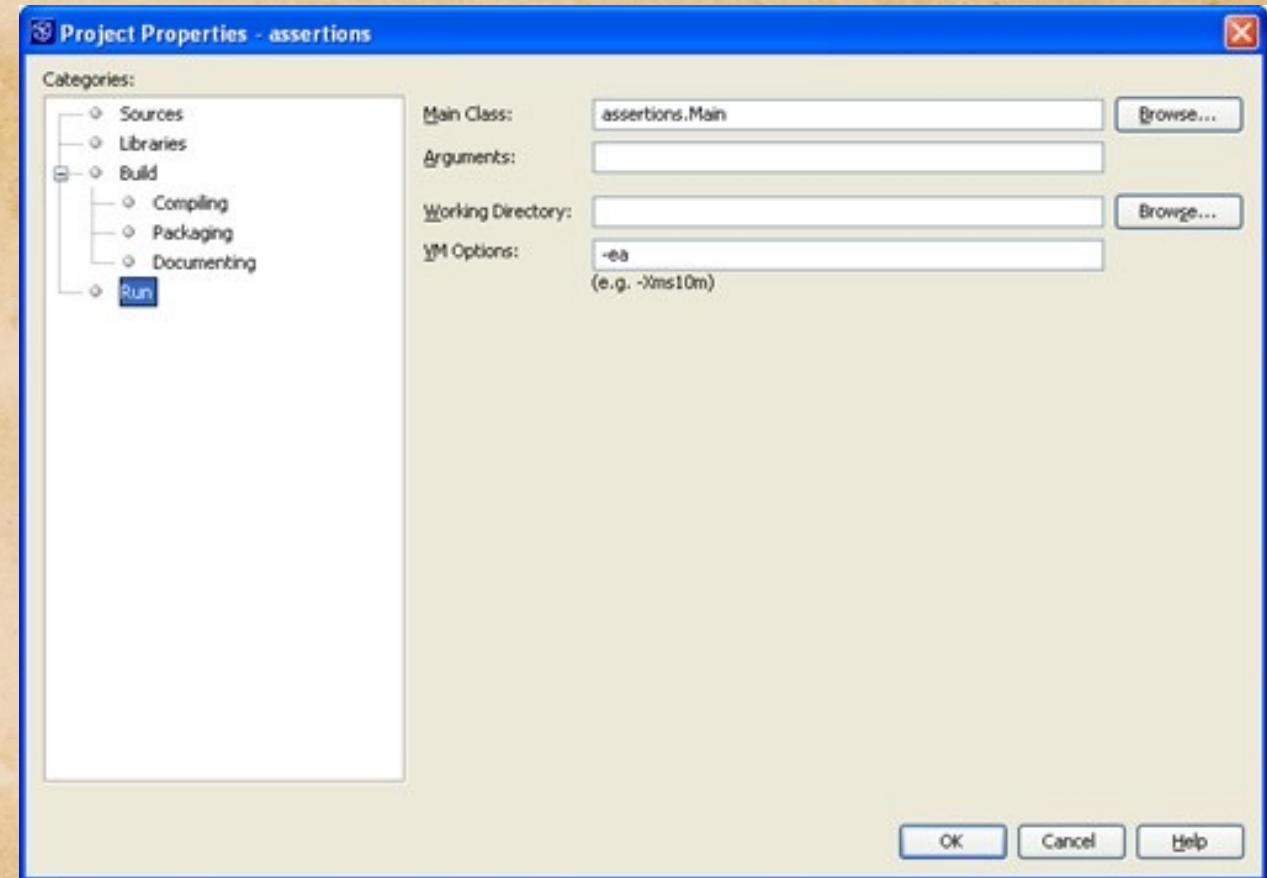
ASSERCIÓNS

- Afirmacions realitzades en un moment particular de l'execució sobre l'estat computacional.
- Si arribaren a ser falses significaria que hi ha algun error en el disseny o utilització de l'algorisme.
- Podem utilitzar la instrucció assert. Està instrucció rep una condició a verificar i, opcionalment, un missatge d'error que retornarà en cas que la condició no es complisca.

ASSERCIÓNS

En Java, les assercions estan desactivades de manera predeterminada durant l'execució.

Han d'habilitar-se explícitament utilitzant l'opció de línia de comandaments -ea (enable assertions) o configurant l'entorn d'execució.



ASSERCIÓNS

- Les assercions no han d'utilitzar-se per a la lògica del programa que afecta directament el flux normal.
- La seua principal funció és verificar suposicions i condicions internes, no com un mecanisme de maneig d'errors.

S'ha de tindre en compte que assert està pensat per a ser usat en l'etapa de desenvolupament i proves. Un programa acabat mai hauria de deixar de funcionar per aquesta mena d'errors.

ASSERCIÓNS

```
[-] public static void main(String[] args) {
    System.out.println(calcularPerimetre(-3));
}

[-] public static double calcularPerimetre(int radi){
    assert radi >= 0 : "El radi no pot ser negatiu";
    return 2 * Math.PI * radi;
}

t - JavaApplication1 (run)

run:
Exception in thread "main" java.lang.AssertionError: El radi no pot ser negatiu
|   at Assercions.calcularPerimetre(Assercions.java:15)
|   at Assercions.main(Assercions.java:12)
```

ASSERCIÓNS

- Precondicions
- Postcondicions

Precondicions

- Són les condicions que han de complir els paràmetres que una funció rep, perquè aquesta es comporte correctament.

```
public int dividir(int dividend, int divisor) {  
    assert divisor != 0 : "El divisor no pot ser zero";  
    return dividend / divisor;  
}
```

Precondicions

- Com a conveni, les precondicions d'un mètode públic en Java és preferible que es comproven mitjançant una condició i llancen l'excepció `IllegalArgumentException` o l'excepció apropiada d'acord amb l'error trobat. No amb l'ús d'assercions.

```
public int dividir(int dividend, int divisor) {  
    if(divisor == 0) throw new IllegalArgumentException("El divisor no pot ser zero");  
    return dividend / divisor;  
}
```

Guard Clauses

- Una clàusula de guarda és simplement una comprovació que ix immediatament de la funció, ja siga amb una instrucció de retorn o amb una excepció.
- Les clàusules de guarda són un senzill mètode que ens permet fer el nostre codi més llegible, més semàntic i amb menor nivell d'indentació.

Guard Clauses

Sense fer ús de Guar Clauses:

```
public void processarDades(String nom, int edat) {  
    // Verificació de condicions sense guard clauses  
    if (nom != null && !nom.isEmpty()) {  
        if (edat >= 0 && edat <= 120) {  
            // Lògica principal del mètode  
            System.out.println("Processant dades...");  
        } else {  
            throw new IllegalArgumentException("L'edat ha d'estar en l'interval de 0 a 120");  
        }  
    } else {  
        throw new IllegalArgumentException("El nom no pot ser nul o buit");  
    }  
}
```

Fent ús de Guard Clauses:

```
public void processarDades(String nom, int edat) {  
    // Guard Clauses amb múltiples condicions  
    if (nom == null || nom.isEmpty()) throw new IllegalArgumentException("El nom no pot ser nul o buit");  
    if (edat < 0 || edat > 120) throw new IllegalArgumentException("L'edat ha d'estar en l'interval de 0 a 120");  
  
    // Lògica principal del mètode després de passar les precondicions  
    System.out.println("Processant dades...");  
}
```

Postcondicions

- Les postcodicions són les condicions que complirà el valor de retorn, (i els paràmetres rebuts, en cas que hagen sigut alterats).

```
public void incrementarTemps(int hores, int minuts, int segons) {  
    segons++;  
  
    if (segons >= 60) {  
        segons = 0;  
        minuts++;  
  
        if (minuts >= 60) {  
            minuts = 0;  
            hores++;  
  
            if (hores >= 24) {  
                hores = 0;  
            }  
        }  
    }  
  
    // Postcondició: Garantir que els segons, minuts i hores estan dins dels límits  
    assert segons >= 0 && segons <= 59 : "Postcondició violada: els segons estan fora del rang permès";  
    assert minuts >= 0 && minuts <= 59 : "Postcondició violada: els minuts estan fora del rang permès";  
    assert hores >= 0 && hores <= 23 : "Postcondició violada: les hores estan fora del rang permès";  
  
    System.out.printf("Temps incrementat: %02d:%02d:%02d\n", hores, minuts, segons);  
}
```