O Control de Erros e o filtrado da entrada de datos.

Nun programa, **os erros e entradas incorrectas son inevitables**. Un usuario pode escribir un valor inesperado, un ficheiro pode non existir, unha rede pode fallar…
Por iso, debemos **controlar os erros** e **filtrar a entrada de datos** para evitar resultados incorrectos, bloqueos ou vulnerabilidades.

Un **Erro** é unha situación non prevista que fai que o programa non poida continuar normalmente ou produza un resultado erróneo ou inesperado.

Unha **Excepción** é un mecanismo polo que un programa pode "notificar" unha situación excepcional. E dicir unha situación na que é necesario realizar un conxunto de accións distinta da habitual, principalmente para ocuparse de resultados erróneos ou inesperados variando o fluxo normal do programa.

Moitos erros se deben a que o algoritmo está traballando con datos que están fora do ámbito previsto, polo que nas aplicacións en produción **é necesario controlar os datos que introducen os usuarios** verificando que sempre sexan correctos.

Control de Erros mediante valores de retorno

O método típico no que un programa controla situacións anómalas durante a execución dunha funcionalidade e mediante valores de retorno especiais fora do dominio dos resultados válidos.

O programador debe comprobar o valor retornado por esa funcionalidade e tomar as medidas necesarias

A principal vantaxe deste método e a sua sinxeleza e eficiencia, pero presenta inconvenientes bastante serios:

- O programador **debe lembrar comprobar sempre os valores de retorno**, o que se omite en moitas ocasións incluso en aplicacións de amplo uso.
- Algunhas veces todos os resultados que pode retornar a funcionalidade corresponden con resultados válidos, e non dispoñemos de ningún valor que poda indicar unha situación de erro de xeito inequívoco.

Cando non dispoñemos de ningún valor que poda servir para indicar unha situación de erro se utilizan mecanismos máis complexos e comprometidos respecto ao tratamento de erros como o uso de variables globais especiais (***errno***) para almacenar os posibles estados de erro.

```
public class ExemploErro {
        public static int dividir(int a, int b) {
            if (b == 0) {
                return -1;  // código de erro
            }
            return a / b;
        }

        public static void main(String[] args) {
            int resultado = dividir(10, 0);
            if (resultado == -1) {
                System.out.println("Erro: división por cero.");
            }
}
```

As Excepcións

Habitualmente en lugar da simple indicación dun erro retornando un valor “especial”, se forza unha situación excepcional (se lanza unha excepción) para obrigar ao programa a tomar as medidas axeitadas (se captura a notificación).

Esta aproximación, a pesar de introducir un pouco de sobrecarga no proceso (menor eficiencia) presenta evidentes vantaxes:

- O programador ten a obriga de tratar o erro, aínda que sexa indicando que decide ignoralo, noutro caso dependendo do tipo de excepción o programa remata ou nin sequera chega a compilar
- O tratamento da situación excepcional queda claramente expresado no código e non rompe a estrutura do programa con continuos condicionais de comprobación.
- E posible usar este mecanismo para indicar situacións “especiais” que non son erros necesariamente.

As Excepcións en Java

En Java dispoñemos dunha clase de obxectos deseñada para isto denominada ***Throwable.*** Un obxecto Throwable pode ser “lanzado” para notificar unha situación especial.

Distinguimos dous grupos de Throwables, os da clase **Error**, e os da clase **Exception**.

Os obxectos da clase Error notifican erros graves que non deben ser capturados, implican a finalización inmediata do programa.

Os obxectos da clase Exception notifican situacións que poden ser tratadas capturando a notificación *(catch)*. Dentro das Exception podemos distinguir entre as “*checked Exception*” e as “*non-checked Exception*”.

As ***non-checked Exception*** poden ser ignoradas, dando lugar a inmediata finalización do fío de execución actual.

As ***checked Exception*** deben ser tratadas obrigatoriamente facendo un tratamento para a situación ou “relanzando” a Exception para que sexa tratada noutra parte do código.

En Java se deben indicar na documentación dos métodos as distintas excepcións que pode lanzar e o significado dos diferentes valores de retorno. No caso das checked Exceptions é obrigatorio indicalo de xeito explícito na cabeceira do método:

```
/**
 * Solicita ao usuario que introduza un número enteiro por consola dentro dun rango específico.
 * * <p>O método repite a solicitude ata que o usuario introduce un valor numérico válido que está dentro
 * do rango [min, max].</p>
 *
 * @param text A mensaxe a mostrar ao usuario para solicitar a entrada.
 * @param min O valor mínimo (inclusive) que o número introducido debe ter.
 * @param max O valor máximo (inclusive) que o número introducido pode ter.
 * @return O número enteiro introducido polo usuario que se atopa no rango especificado.
 * @throws IllegalArgumentException se o rango especificado non é válido (se max é menor que min).
 * @throws InterruptedException se usuario desexa cancelar a operación de entrada.
 */
public static int inputInteger(String text, int min, int max)   throws IllegalArgumentException, InterruptedException {
    // ...
}
```

Neste exemplo, se indica de xeito explícito que o método pode lanzar as checked-Exceptions IllegalArgumentException ou InterruptedException. O código que utilice este método estará obrigado a capturar e tratar estas excepcións (catch) ou a relanzalas a súa vez ao método anterior.

Si todos os métodos relanzan as excepcións (throws) en lugar de tratalas, o sinal chegará ao método principal do programa (main) e levará a cabo o tratamento especificado ou finalizará o programa.

As “sinais” que se lanzan son obxectos dunha clase que sexa Throwable (descendentes de Throwable ou Error) que é necesario crear, e que se lanzan coa instrucción **throws**

A Captura e Tratamento de Excepcións: Os bloques `try-catch-finally`

Como vimos, cando se produce unha situación excepcional, un programa pode lanzar un obxecto Throwable para enviar un "sinal". Para evitar que este sinal cause a finalización abrupta do noso programa, debemos "capturalo" e darlle un tratamento adecuado. Isto conséguese mediante os bloques try, catch e finally.

```
// Indicamos con throws que o método pode notificar a situación InterruptedException
public static int inputInteger(String text, int min, int max) throws InterruptedException {
    // Inicializamos o escáner para ler a entrada do usuario desde a consola.
    Scanner scn = new Scanner(System.in);
    // Declaramos a variable para almacenar o número.
    // Debemos inicializalo, porque o compilador “non está seguro” de que non quede sen inicializar
     int number=0;
    // Usamos unha variable booleana para controlar o bucle.
    // O bucle seguirá executándose mentres "ok" sexa falso.
    String input;
    boolean ok = false;
    // Bucle principal para solicitar a entrada ata que o usuario
    // introduza un valor válido ou cancele a operación.
    while (!ok) {
        try {
            // 1. O programa pide a entrada do usuario.
            System.out.print(text);
            input = scn.nextLine();
            // 2. Comprobamos se o usuario quere cancelar.
            // Se a entrada é "*", lanzamos unha excepción para deter a operación.
            if (input.equals("*")) {
                throw new InterruptedException("Operación cancelada");
            }
            // 3. Convertemos a cadea de texto nun número enteiro.
            // Se o usuario escribe "abc", parseInt lanzará unha NumberFormatException.
            number = Integer.parseInt(input);
            // 4. Se a conversión foi exitosa, comprobamos o rango.
            // Se está fóra do rango, lanzamos outra excepción.
            if ((number < min) || (number > max)) {
                throw new IllegalArgumentException("O valor debe estar entre " + min + " e " + max);
            }
            // 5. Se chegamos ata aquí, todo foi ben.
            // A variable "ok" ponse en verdadeiro para saír do bucle.
            ok = true;
        } catch (NumberFormatException error) {
            // Este bloque captura a excepción se o usuario non introduce un número.
            System.out.println("Debes escribir un número enteiro.");
        } catch (IllegalArgumentException error) {
            // Este bloque captura a excepción se o número está fóra do rango.
            // Visualizamos a mensaxe especificada cando lanzamos (throw) a excepción
            System.out.println(error.getMessage());
        }
        // Nota: As excepcións InterruptedException teñen que ser tratadas
        // polo método que chama a este, xa que o "throws" na cabeceira o indica e non a capturamos con catch.
    }
    // Devolvemos o número válido.
    return number;
}
```

O bloque ***try*** serve para encerrar o código que consideramos "arriscado", é dicir, o código que pode lanzar unha excepción si se deran as circunstancias. O programa tentará executar todas as instrucións dentro deste bloque. Se todo vai ben, a execución continúa de forma normal despois dos bloques ***catch*** e ***finally***.

O método anterior podería utilizarse do seguinte modo:

```
import java.time.Year;

public class App {

    public static void main(String[] args) {
        System.out.println("--- Calculadora de ano de nacemento ---");

        int idade = 0;

        // O bloque 'try' encerra o código que pode lanzar excepcións.
        try {
            // Chamamos ao noso método 'inputInteger'.
            // Esta chamada pode lanzar a excepción  'InterruptedException'.
            // O método insistirá na entrada ata que o usuario introduza un valor válido ou desista de facer a operación
            idade = inputInteger("Introduce a túa idade: ", 1, 120);

            // Se a execución chega aquí, non se lanzou ningunha excepción
            // e a variable 'idade' contén un valor válido.
            int anoActual = Year.now().getValue();
            int anoNacemento = anoActual - idade;
            System.out.println("Naciches aproximadamente no ano: " + anoNacemento);

        }  catch (InterruptedException e) {
            // Este bloque 'catch' execútase se o usuario cancela a operación.
            System.out.println("A operación foi cancelada.");

        } finally {
            // O bloque 'finally' execútase sempre, tanto se se lanza
            // unha excepción como se non. É ideal para tarefas de limpeza e liberación de recursos
            System.out.println("\nProceso de entrada de datos finalizado.");
        }
        System.out.println("O programa rematou.");
    }
```

Neste código podemos observar:

1. **Cadea de chamadas (propagación da excepción)**

   Cando unha excepción ocorre nun método e non se captura, propágase ao método que chamou a ese método, e así sucesivamente ata chegar ao main.
   Isto permite centralizar o tratamento en niveis superiores do programa, pero tamén hai que coñecer o fluxo para evitar erros inesperados.

2. **Finally e recursos**

   O bloque finally sempre se executa independente de que haxa excepción ou non. É útil para liberar recursos (ficheiros, conexións, sockets) incluso cando ocorre un fallo.

3. **Relanzamento de excepcións**

   Unha excepción capturada pódese relanzar para que sexa tratada noutro nivel do programa. Permite modularidade e non perder información da excepción orixinal.