Material descargable

egg

Teoría JAVA I

Bloque try-catch

El **bloque "try-catch"** en Java se emplea para manejar excepciones. El código propenso a errores o que puede generar excepciones se coloca dentro del bloque "try". Si ocurre una excepción dentro de este bloque, se captura en uno o varios bloques "catch", donde se especifica el tipo de excepción que se desea manejar y el código correspondiente para tratar la excepción.

Para comprender su funcionamiento, analicemos los siguientes dos ejemplos y observemos lo que se muestra por consola:

```java
public static void main(String[] args) {
        Scanner pepe = new Scanner(System.in);
        System.out.print("Ingrese un número divisor de 10: ");
        int numero = pepe.nextInt();
        try {
            double resultado = 10/numero;
            // Posible división por cero
            System.out.println("El resultado es: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: No es posible dividir por 0.");
        }
        System.out.println("¡Gracias! ");
}
```

```
CONSOLA:
Ingrese un número divisor de 10: 0
Error: No es posible dividir por 0.
¡Gracias!
```

```java
public static void main(String[] args) {
       Scanner pepe = new Scanner(System.in);
        System.out.print("Ingrese un número divisor de 10: ");
        int numero = pepe.nextInt();
        double resultado = 10/numero;
        // Posible división por cero
        System.out.println("El resultado es: " + resultado);
        System.out.println("Gracias! ");
}
```

```
CONSOLA:
Ingrese un número divisor de 10: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at EjemplosTeoria.main(EjemplosTeoria.java:
```

	8)

En los dos ejemplos anteriores, se puede observar que sin el *"bloque try-catch"*, nuestro programa se detiene en la línea que genera el error *"double resultado = 10/numero;"*, impidiendo que la ejecución continúe.

En contraste, al utilizar el *"bloque try-catch"*, el programa salta las líneas de código dentro del *"bloque try"* cuando se encuentra con una línea que produce un error. En lugar de detenerse, continúa ejecutando el programa a partir del *"bloque catch"*, permitiendo que las líneas de código restantes se ejecuten después del bloque.

Manejo de excepciones

En Java, **existen diversos tipos de excepciones y es posible declarar diferentes *bloques catch* para capturarlas** y realizar acciones específicas para cada una de ellas.

En el siguiente ejemplo de código, ilustramos este concepto:

```
public static void main(String[] args) {
        try {
            Scanner pepe = new Scanner(System.in);
            System.out.print("Ingrese un divisor: ");
            int numero = pepe.nextInt();// Posible entrada inválida
            String palabra = "hola";
            double resultado = 10 / numero ;// Posible división por cero
            System.out.println("El resultado es: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("Error: División por cero.");
        } catch (InputMismatchException e) {
            System.out.println("Error: Se detectó un valor inválido ingresado
por teclado.");
        } catch (Exception e) {
            System.out.println("Error: Ups!");
        }
 }
```

El *"InputMismatchException"* ocurre cuando se produce un error al intentar convertir la entrada del usuario a un tipo de dato de Java, por ejemplo, si el usuario ingresa letras cuando se espera un número.

En el ejemplo anterior, se utiliza *"Exception"* como un bloque catch adicional que atrapa cualquier excepción no especificada en los bloques catch anteriores.

Es importante colocar el *bloque catch de Exception* al final, ya que de lo contrario, este bloque interceptaría cualquier excepción antes de que los bloques catch específicos pudieran ser utilizados.

```
public static void main(String[] args) {
        try {
            // Código que puede lanzar una excepción
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error: Ups!: " + e.getMessage());
        }
 }
```

Dentro de los bloques catch, podemos observar la *variable "e"*, que es del tipo de la excepción capturada. *Esta variable nos proporciona acceso a métodos útiles que podemos utilizar.*

A continuación, veremos dos de ellos:

- **printStackTrace():** Este método imprime por consola la *"pila de llamadas de la excepción"*, mostrando la información sobre el lugar exacto donde ocurrió el problema. Esto resulta especialmente útil en aplicaciones más complejas, ya que nos permite identificar y solucionar errores de manera más efectiva. Por ejemplo:

```
java.util.InputMismatchException
      at java.base/java.util.Scanner.throwFor(Scanner.java:947)
      at java.base/java.util.Scanner.next(Scanner.java:1602)
      at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
      at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
      at EjemploTeoria.main(EjemploTeoria.java:9)
```

  En la imagen podemos observar la salida de un *"printStackTrace()"* dentro de un *bloque catch* que captura una *Excepción "InputMismatchException"*.

Destacado en azul, se puede ver que se menciona la clase *"EjemploTeoria"* donde ocurre la excepción, así como el método main y la línea de código número 9 del archivo.

- **getMessage():** Este método devuelve únicamente el mensaje de descripción del error, sin incluir la pila de llamadas. Puede ser utilizado para complementar mensajes personalizados de error. En futuros temas, exploraremos otros usos que se le pueden dar a este método.

Algunas excepciones utilizadas con frecuencia

- **NullPointerException**: Esta excepción se produce cuando se intenta acceder a un objeto que tiene un valor null. Por ejemplo, si intentas llamar a un método o acceder a un atributo de un objeto que no ha sido inicializado.
- **ArrayIndexOutOfBoundsException**: Ocurre cuando se intenta acceder a un índice fuera del rango válido de un array. Por ejemplo, al intentar acceder a un índice negativo o mayor que el tamaño del array.
- **ArithmeticException**: Esta excepción se produce cuando se intenta realizar una operación aritmética inválida, como dividir por cero.
- **NumberFormatException**: Se lanza cuando se intenta convertir una cadena a un tipo numérico, pero la cadena no tiene el formato adecuado para ello.
- **FileNotFoundException**: Se produce cuando se intenta acceder a un archivo que no existe en el sistema de archivos.
- **IOException**: Esta es una excepción genérica que se lanza cuando ocurre un error de entrada/salida durante la lectura o escritura de datos.
- **ClassNotFoundException**: Ocurre cuando se intenta cargar dinámicamente una clase que no se encuentra en el classpath.
- **InterruptedException**: Esta excepción se produce cuando un hilo en ejecución es interrumpido por otro hilo mientras está esperando, durmiendo o realizando alguna operación de bloqueo.

💡 En la documentación oficial, podrás conocer los tipos de excepciones existentes, te **invitamos a acceder** para investigar sobre las mismas.

Ámbito de las variables

En Java, **las variables que se declaran dentro de un bloque tienen un ámbito limitado a ese bloque en particular**. Esto implica que solo son visibles y accesibles dentro de dicho bloque y no se pueden acceder desde fuera de él. Una vez que se sale del bloque, esas variables dejan de existir y no se pueden utilizar.

💡Recuerda que un bloque de código es aquel que se encuentra entre llaves "{}".

Hasta ahora, esto no nos había afectado porque trabajábamos exclusivamente dentro del bloque de código del método main. Sin embargo, *con la incorporación de estructuras de control, surgen nuevos bloques de código, por lo tanto, debemos tener cuidado al declarar nuestras variables* si posteriormente deseamos utilizarlas fuera de los bloques que creamos.

Observemos los siguientes ejemplos:

```java
public static void main(String[] args) {
        int dato1 = pepe.nextInt();
        int dato2 = pepe.nextInt();
        if (dato1 != 0 && dato2 != 0) {
            int resultado = dato1 + dato2;
        }
        System.out.println(resultado);
}
```

```java
public static void main(String[] args) {
        int dato1 = pepe.nextInt();
        int dato2 = pepe.nextInt();
        int resultado;
        if (dato1 != 0 && dato2 != 0) {
            resultado = dato1 + dato2;
        }
        System.out.println(resultado);
}
```

La línea System.out.println(resultado); marca un error porque la variable "resultado" no existe dentro del ámbito del método main. Se declara dentro del bloque if y no se puede acceder a ella fuera de ese bloque.	El programa se compila correctamente porque la variable "resultado" se declara dentro del ámbito del método main. De esta manera, se puede acceder a ella fuera del bloque if.
