Programación orientada a objetos

Manejo de excepciones

{desafío} latam_

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje Java para resolver un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Comprender las excepciones de Java para optimizar nuestras aplicaciones.*
- *Aplicar Excepciones utilizando Try-Catch y "throw" para controlar y capturar las excepciones.*

¿Qué entendemos por excepción?

/* Excepciones en Java */

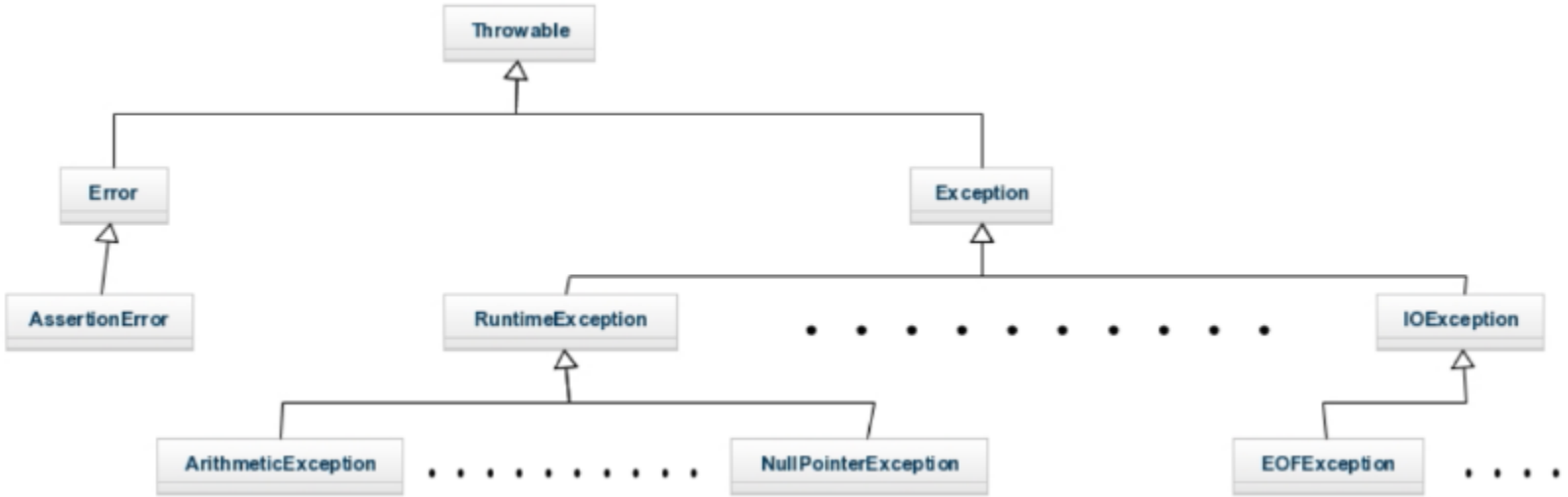
Excepciones

- Control de errores en ejecución del programa; esto quiere decir que las excepciones nos permiten controlar errores de usuarios o de programación.
- Tienen jerarquía entre ellas.

Las excepciones más utilizadas en programación son las de tipo aritméticas, de validaciones de nulos y de archivos para controlar los errores.

Excepciones

Ilustración



Throwable

Representa todo lo que Java puede “lanzar”, de hecho la palabra throwable se puede considerar en el español como “lanzar o arrojar”. A su vez esta clase:

- Almacena un mensaje (variable de instancia de tipo String) que podemos utilizar para detallar qué error se produjo.

- Puede ser una causa, también de tipo Throwable, que permite representar el error que causó este error.

Clases que nos permiten controlar errores y excepciones

Error

Indica que el error es a nivel de hardware y no aplicativo,

Exception

Manejamos la mayor cantidad de errores controlables del software.

- RuntimeException:
- IOException:

Principales excepciones

- RuntimeException

Errores del programador, como una división por cero o el acceso fuera de los límites de un array.

- IOException

Errores que no puede evitar el programador, generalmente relacionados con la entrada/salida del programa.

Métodos listos de Java para las excepciones

- getMessage()

Permite mostrar el error y dónde está pasando, esto quiere decir, la línea de la clase de Java o el tipo de error.

- printTrace()

Permite mostrar el error con más detalle, sin embargo, utilizarlo en todo el código nos provoca un mayor uso de la memoria de la JVM, lo cual es recomendable utilizarlo cuando hay un error en particular que no se logra detectar.

Excepciones personalizadas

Java nos ofrece su clase Exception para su uso, pero también nos ofrece heredar de esta clase para crear nuestras propias clases con nuestros propios métodos.

Esto nos permite controlar las excepciones según funcionalidad, evento o lógica en el programa.

Excepciones personalizadas

En este ejemplo tenemos el uso de Herencia y la clase MiExcepcion que hereda de Exception, sobreescribimos el constructor y creamos un método llamado “validaNulo”, el cual valida si el campo es nulo.

```
public class MiExcepcion extends Exception {
     public MiExcepcion(String arg) {
          super(arg);
     }
     public String validaNulo(String arg) {
          String mensaje ="";
          if(arg == null) {
               mensaje= "campo nulo";
          }
          return mensaje;
     }
}
```

/* Try - Catch */

Try - Catch

Try

- Código que debe funcionar sin problemas. Podemos hacer llamadas a otras clases, validaciones y escribir todo el código que debiese funcionar.
- Esta palabra va acompañada con el uso de llaves de apertura y cerrado, y siempre se utiliza con la otra palabra reservada de Java Catch.

Catch

- Lanzamos las posibles excepciones del bloque de código dentro del try.
- Pueden existir tantos catch como Exception se quiera controlar, esto quiere decir que para utilizar Catch debemos siempre utilizar el nombre de la Excepción dentro del paréntesis, al lado de la palabra, y, a continuación, abrir y cerrar bloque de código con el uso de llaves como se muestra en la imagen a continuación.

Ejemplo

División por 0

```
try {
    int total = 3 / 0;
}catch(Exception excepcion) {
    int total = 0;
}
```

Ejemplo

Se puede utilizar más de un catch a la vez, por ejemplo, si sabemos qué excepciones podría arrojar un bloque try, podemos agregar un catch con cada subclase de Exception que consideremos necesaria.

```
try {
      Scanner sc = new Scanner(System.in);
      String variable = sc.nextLine();
      if(variable.isEmpty()){
            variable = null;
      }
      int total = 3 / Integer.parseInt(variable);
}catch(NullPointerException ex1) {
      System.out.println("No se puede dividir por un valor nulo.");
      int total = 0;
}catch(NumberFormatException ex2) {
System.out.println("El valor de variable no es un número.");
int total = 0;
}catch(Exception ex3){
      System.out.println("Error inesperado: "+ex3.getMessage());
      int total = 0;
}
```

/* La cláusula finally */

Cláusula finally

Ejecutar un fragmento de código independientemente de si se produce o no una excepción.

Se utiliza, por ejemplo, cuando se debe cerrar un fichero que estemos manipulando o cerrar una conexión de base de datos para liberar recursos.

```
Connection cn = null;
try {
      cn = fuenteDeDatos.getConnection();
}catch(Exception e){
      System.err.out("Ha ocurrido un error");
}finally {
      cn.close();
}
```

Cláusula finally

Otro ejemplo con código

```
public void validaEdad(String arg) {
     String mensaje =”prueba”
     try {
          if((Integer.parseInt(arg)) >=18){
               System.out.println("Edad es mayor a 18 y un número" + mensaje);
          }
     }catch (NumberFormatException e) {
          System.out.println(e.getMessage() + mensaje);
     }
}
```

Cabe mencionar que cada variable que se escriba dentro de los bloques de llave de apertura y cerrado, solo será vista y podrá utilizarse ahí. Se recomienda que al utilizar variables se declaren antes del bloque try-catch.

Throw

Para lanzar un objeto de tipo Exception se necesita utilizar la palabra reservada de Java “Throw”, esta nos permite lanzar exception en ejecución y capturar la ejecución de un objeto. Cuando se lanza una excepción:

1. Se sale inmediatamente del bloque de código actual.
2. Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepción generada, se ejecuta el cuerpo de la cláusula catch.
3. Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.
4. El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecución con un mensaje de error.

Throw

```
public void validaEdad(String arg) {
    try {
        if((Integer.parseInt(arg)) >=18){
            System.out.println("Edad es mayor a 18");
        }
    }catch (NumberFormatException e) {
        throw new NumberFormatException(e.getMessage());
    }
}
```

Ejercicio guiado

Validaciones con Try-Catch

Crear un método que nos permita dividir dos parámetros de tipo String y retornar el resultado de la división.

- Paso 1: Crear la variable local de método resultado.
- Paso 2: Realizar la conversión de datos de String a Int dentro del bloque try.
- Paso 3: Utilizar el primer catch con la Exception NumberFormatException, esta controlará la excepción cuando uno de los dos parámetros sea algo distinto a un número.

  ```
  java.lang.NumberFormatException: Formato de número incorrecto :For input string: "dos"
  ```
- Paso 4: Utilizar el segundo Catch con la Exception ArithmeticException. Esta controlará la Exception cuando uno de los dos parámetros sea cero, dando la división de una excepción de tipo aritmética, por ejemplo 0/2.

  ```
  java.lang.ArithmeticException: Error en aritmetico : / by zero
  ```

Validaciones con Try-Catch

Crear un método que nos permita dividir dos parámetros de tipo String y retornar el resultado de la división.

- Paso 5: Si no entra a ninguna Exception, retorna el resultado.

```
public static void main (String [] args) {
	division("22","0");
}
public static int division(String valorUno, String valorDos) {
	int resultado = 0;
	try {
		int uno = Integer.parseInt(valorUno);
		int dos = Integer.parseInt(valorDos);
		resultado = uno/dos;
	}catch (NumberFormatException e) {
		//se lanza cuando el parámetro sea distinto a una numero throw new
	NumberFormatException("Formato de número
		incorrecto :" + e.getMessage());
	}catch (ArithmeticException e) {
		// se lanzará cuando el parámetro sea un cero throw new
	ArithmeticException("Error en aritmética :
		" +e.getMessage());
	}
	return resultado;
}
```

¿A qué clase corresponde la siguiente definición?

“Es una clase base que representa todo lo que Java puede ‘`lanzar`’”

¿Qué implica una excepción del tipo “RunTimeException”?

Próxima sesión...

- *Desafío evaluado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales