Módulo 1. Manejo de excepciones en Java

CÓRDOBA CREA FUTURO

Introducción

El manejo de excepciones es una técnica fundamental en programación para gestionar errores imprevistos o situaciones anómalas que ocurren durante la ejecución de un programa. Permite que el código capture errores (por ejemplo, divisiones por cero o archivos no encontrados) y actúe en consecuencia, en lugar de cerrarse de forma inesperada, lo que mejora la fiabilidad.

Las excepciones son el mecanismo mediante el cual los programas de Java gestionan los diferentes errores que pueden ocurrir. Cuando una excepción ocurre, se dice que fue «lanzada»; cuando se la maneja, es decir, cuando se realiza alguna acción frente al error, se dice que fue «capturada».

Importancia de las excepciones

Las excepciones son elementos fundamentales en la construcción de programas robustos y fiables en cualquier

lenguaje de programación. Su importancia radica en varios aspectos. A continuación, se desarrollan algunos de ellos:

- **Manejo de errores de manera estructurada.** Las excepciones brindan una forma estructurada de manejar errores y situaciones excepcionales. Esto permite separar el código de manejo de errores del código funcional, lo que da como resultado un código más limpio y fácil de entender.

- **Mejora de la robustez y confiabilidad.** Al anticipar y manejar situaciones excepcionales de forma adecuada, se logra que las aplicaciones sean más resistentes y confiables. Esto permite que el programa funcione de manera predecible y sin interrupciones.

- **Retroalimentación al usuario.** Mediante el uso de excepciones, es posible proporcionar mensajes de error significativos y detallados al usuario, lo que facilita la comprensión de lo que salió mal y cómo corregirlo. De este modo, se mejora la experiencia del usuario y se facilita la resolución de problemas.

- **Flexibilidad y adaptabilidad.** El manejo adecuado de excepciones permite adaptar el comportamiento de la aplicación según las circunstancias. Esto implica que se puede modificar la forma en que se gestionan las excepciones en distintas partes del programa para ajustarse a los requisitos específicos de cada sección.

CONTINUAR

1. Tipos de excepciones en Java

Las excepciones se dividen en tres categorías principales, cada una heredada de la clase Throwable. Estas categorías son las siguientes:

- *Checked exceptions* **(excepciones comprobadas).** Son excepciones que deben declararse en la firma del método o capturarse de forma explícita en un bloque `try-catch`, o bien declararse con *throws*. Si una excepción comprobada no se maneja correctamente, el código no se compila; es decir, se verifican en tiempo de compilación. Estas excepciones heredan de la clase Exception (por ejemplo, `IOException`, `SQLException`).

- *Unchecked exceptions* **(excepciones no comprobadas).** Son excepciones que no están obligadas a manejarse de forma explícita.

Ocurren durante la ejecución del programa y no se requiere que se declaren en la firma del método ni que se capturen mediante un bloque *try-catch*. Heredan de la clase RuntimeException (por ejemplo, `NullPointerException`, `ArithmeticException`).

- ***Errors.*** Son problemas graves que, por lo general, están fuera del control del programador y no deben manejarse de forma explícita. Indican situaciones serias que deberían detener la ejecución del programa. Heredan de la clase Error (por ejemplo, `OutOfMemoryError`).

ASPECTOS CLAVE | CAPTURANDO EXCEPCIONES

A continuación, se presentan algunos aspectos relacionados con el manejo de excepciones:

- **Estructura** `try-catch`. El código se divide en un bloque *try* (donde puede ocurrir el error) y un bloque *catch* (donde se maneja la excepción).
- **Enfoque preventivo**: ayuda a evitar que el sistema falle por completo ante situaciones excepcionales.
- **Control del flujo**: permite que el programa continúe funcionando, muestre un mensaje de error personalizado o se recupere ante la situación.

Componentes fundamentales

En el siguiente listado se describen los principales componentes del manejo de excepciones:

- Try. Define un bloque de código propenso a errores que se intenta ejecutar.
- Catch: captura y maneja la excepción específica si ocurre dentro del bloque *try*.
- Finally: ejecuta un bloque de código siempre, se haya producido o no una excepción; resulta adecuado para liberar recursos.
- Throw: lanza de forma explícita una excepción.
- Throws: se utiliza en la firma del método para declarar las excepciones que este puede lanzar.

ASPECTOS CLAVE | **CAPTURANDO EXCEPCIONES**

Para capturar excepciones en un bloque de código susceptible, como se mencionó, se debe utilizar un bloque try-catch, que consiste en un bloque de instrucciones en el que pueden producirse excepciones (bloque try) y en uno o más bloques destinados a su manejo (bloques catch). El formato es el siguiente:

```
try {
  // sentencias a monitorear el error
}
catch (tipoexcepcion nombrevar) {
  // sentencias de manejo de la excepción
```

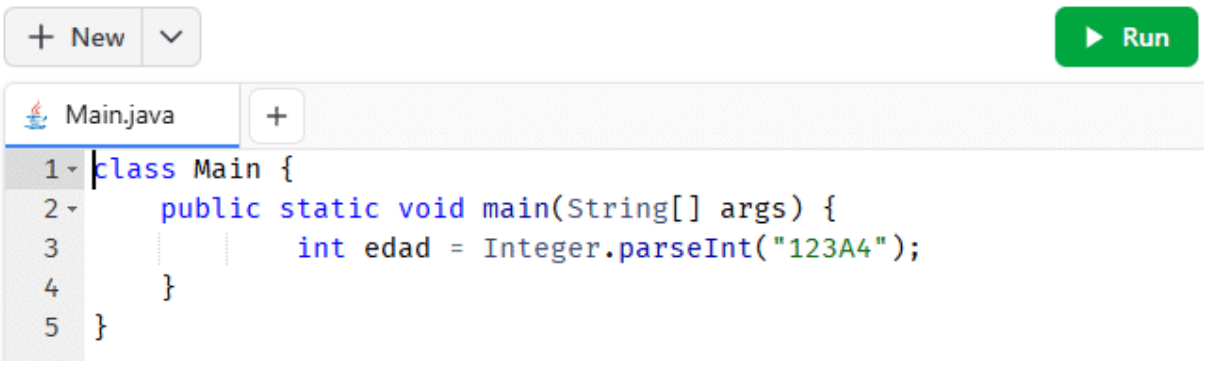
```
}
finally {
 //sentencias a ejecutar ocurran o no excepciones
}
```

Para comprender el concepto de excepciones, se presentarán dos ejemplos: uno mediante código en lenguaje Java sin manejo de excepciones y otro con su correspondiente manejo. Ambos se aplicarán a la conversión de un dato, como se muestra a continuación: convertir la cadena «123A4» en número.

Sin excepción

Haz clic en la imagen y comprueba qué sucede al ejecutar el código:

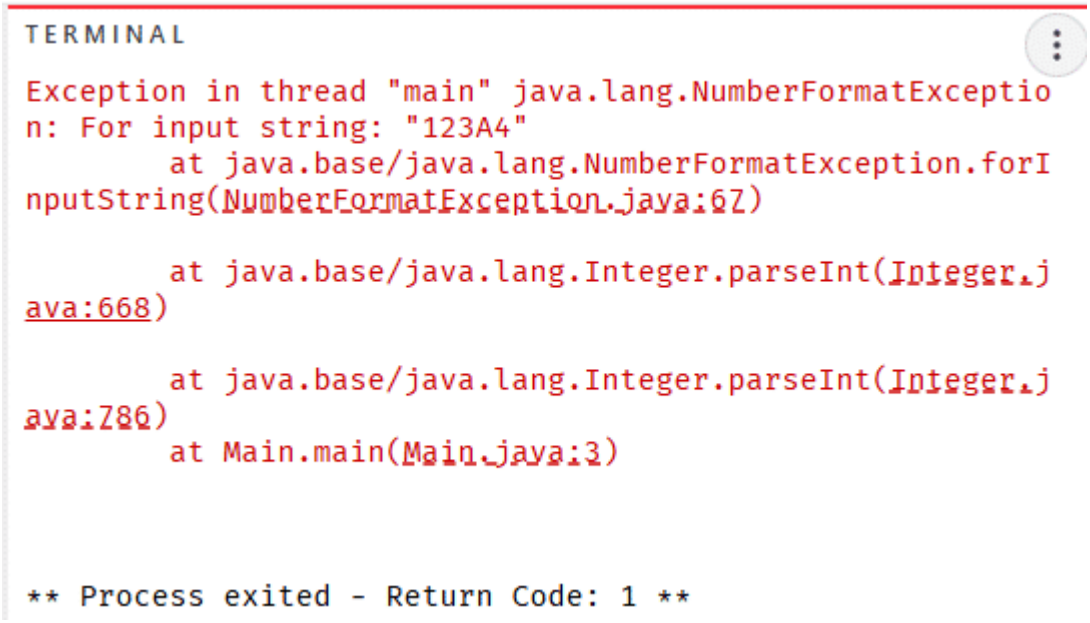
Figura 1. Código en Java sin manejo de excepciones



Fuente: captura de pantalla de Online-Java (www.online-java.com)

La ejecución del código arrojará el siguiente resultado:

Figura 2. Error de conversión sin manejo de excepciones

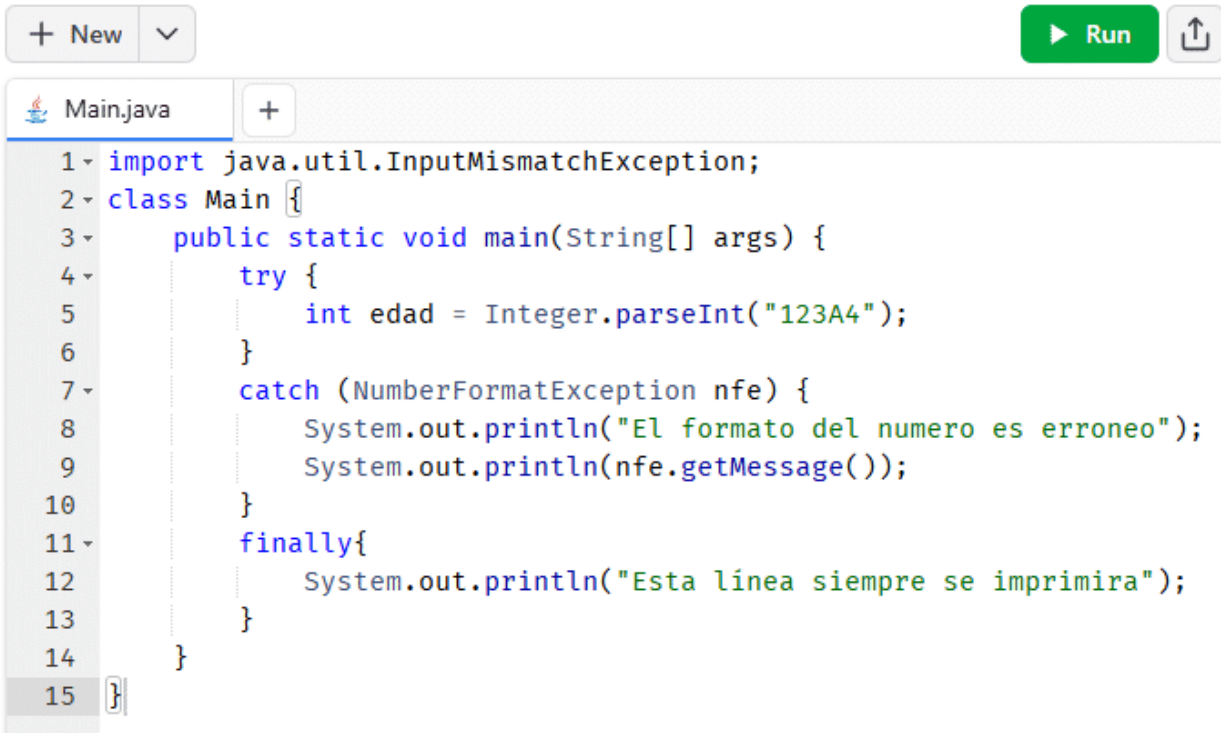


Fuente: captura de pantalla de Online-Java (www.online-java.com)

Con excepción

El siguiente ejemplo muestra la ejecución del programa; en este caso se maneja la excepción, de modo que el error se comunica al usuario sin que el programa deje de ejecutarse de forma abrupta. Haz clic en la imagen para comprobar su funcionamiento.

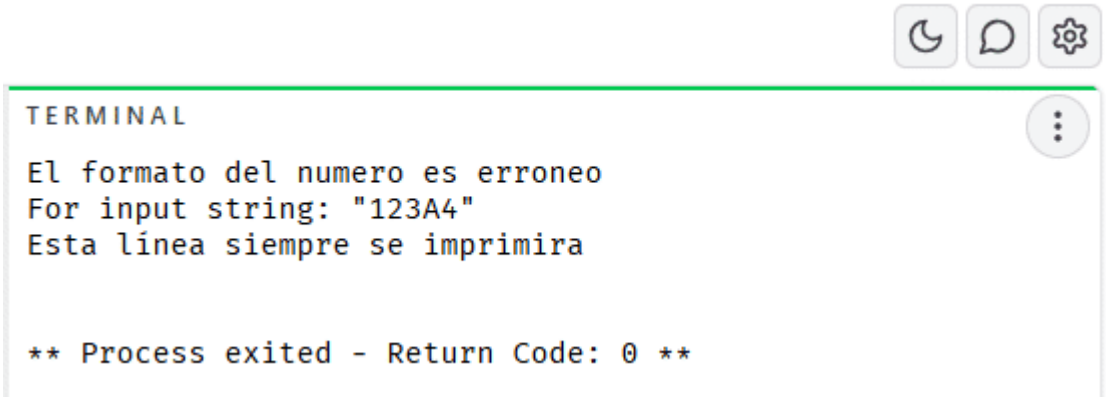
Figura 3. Código en Java con manejo de excepciones



Fuente: captura de pantalla de Online-Java (www.online-java.com) https://www.online-java.com/Lk6qInjmht

Si se ejecuta el programa presionando «Run», se obtiene el siguiente mensaje:

Figura 4. Resultado del programa con manejo de excepciones



Fuente: captura de pantalla de Online-Java (www.online-java.com)

Como se puede observar en la imagen 4, siempre se imprime el mensaje «Esta línea siempre se imprimirá».

En otro ejemplo, se muestra un error por división por cero que es capturado e informado al usuario que ejecuta el programa:

```
try {
```

```
            int resultado = 10 / 0; // Provoca ArithmeticException

        } catch (ArithmeticException e) {

          System.out.println("Error: División por cero.");

        } finally {

            System.out.println("Este bloque siempre se ejecuta.");

        }
```

Buenas prácticas

En el siguiente listado se presentan algunas recomendaciones para el manejo de excepciones:

- **No silencies excepciones**. Evita bloques `catch` vacíos.
- **Captura excepciones específicas** en lugar de utilizar la clase genérica Exception.

- **Utiliza** `finally` para cerrar conexiones o archivos.

Puedes practicar en el sitio web W3Schools, que también ofrece la posibilidad de trabajar con ejemplos propios: https://www.w3schools.com/java/java_try_catch.asp.

Excepción no capturada

Cuando en un programa se lanza una excepción y esta no se captura, supera los límites del programa y es interceptada por la Java Virtual Machine (JVM), que muestra un mensaje similar al siguiente:

```
Exception in thread "main“

java.lang.NullPointerException

at MiClase.main(MiClase.java:17)
```

CLASES DE EXCEPCIONES EN JAVA	EJEMPLOS CON CLASES DE EXCEPCIONES	CREACIÓN DE EXCEPCIONES PERSONALIZADAS

A continuación, se presentan algunas de las clases de excepciones más comunes según su categoría:

1. *Checked exceptions*

- **IOException**: indica un error en una operación de entrada o salida (E/S).
- **FileNotFoundException**: se lanza cuando se intenta acceder a un archivo que no se encuentra en el sistema.
- **ParseException**: ocurre al intentar analizar una cadena para convertirla en un formato específico.
- **SQLException**: representa una excepción específica que se produce durante la interacción con una base de datos.

2. *Unchecked exceptions*

- **RuntimeException**: clase base para excepciones no comprobadas.
- **NullPointerException**: ocurre cuando se intenta acceder a un objeto cuyo valor es null.
- **ArrayIndexOutOfBoundsException**: se lanza cuando se intenta acceder a una posición fuera del rango válido de un arreglo.
- **ArithmeticException**: indica un error aritmético, como la división por cero.

Además de estas, existen muchas otras excepciones específicas para situaciones particulares. Es importante conocerlas y comprender cuándo y cómo manejarlas para escribir un código robusto y confiable.

A continuación, se presentan ejemplos de cada tipo de excepción en Java. Comencemos con las *checked exceptions*:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class CheckedExample {
  public static void main(String[] args) {
    try {
      File file = new File("archivo.txt");
      FileReader fileReader = new FileReader(file);
    } catch (FileNotFoundException e) {
      System.out.println("El archivo no se encontró.");
    }
  }
}
```

En este caso, el código intenta abrir un archivo llamado «archivo.txt» para su lectura mediante un `FileReader`. Si el archivo no se encuentra, se captura la excepción `FileNotFoundException` y se imprime un mensaje indicando que el archivo no fue hallado.
Ahora, veamos un ejemplo de *unchecked exceptions*:

```
public class UncheckedExample {
  public static void main(String[] args) {
    int[] arr = {1, 2, 3};
    System.out.println(arr[5]);  // Genera ArrayIndexOutOfBoundsException
  }
```

```
}
```

Este código intenta imprimir el valor en la posición 5 del arreglo. Sin embargo, el arreglo solo contiene los índices 0, 1 y 2; por lo tanto, al intentar acceder al índice 5 se genera una excepción `ArrayIndexOutOfBoundsException`, ya que se encuentra fuera del rango válido.

La excepción `ArrayIndexOutOfBoundsException` es una excepción no comprobada en Java, lo que significa que no es obligatorio manejarla de forma explícita mediante un bloque *try-catch*. Esto se debe a que es responsabilidad del programador garantizar que se acceda a los elementos del arreglo dentro de sus límites válidos.

La creación de excepciones personalizadas en Java resulta útil cuando es necesario representar situaciones excepcionales específicas de una aplicación que no están contempladas por las excepciones predefinidas. Esto permite construir jerarquías de excepciones que reflejen de manera más precisa las situaciones inusuales que pueden presentarse.

A continuación, se muestra cómo crear una excepción personalizada en Java:

```
public class MiExcepcionPersonalizada extends Exception {
  public MiExcepcionPersonalizada(String mensaje) {
     super(mensaje);
  }
}
```

En este ejemplo, se crea una clase llamada MiExcepcionPersonalizada que extiende de Exception. Además, se define un constructor que recibe un mensaje como argumento y lo pasa al constructor de la clase base Exception mediante super(mensaje).

Seguidamente, se presenta un ejemplo de cómo lanzar y capturar esta excepción:

```
public class Ejemplo {
  public static void main(String[] args) {
      try {
          throw new MiExcepcionPersonalizada("Este es un mensaje de excepción personalizada.");
      } catch (MiExcepcionPersonalizada e) {
                    System.out.println("Se ha producido una excepción personalizada: " + e.getMessage());
      }
  }
}
```

En el método *main*, se lanza una instancia de MiExcepcionPersonalizada mediante la palabra reservada throw. Posteriormente, se captura esta excepción utilizando un bloque *try-catch* y se imprime el mensaje asociado a la excepción.

¿Cuándo es útil crear excepciones personalizadas?

Crear excepciones personalizadas resulta útil en diversos escenarios al desarrollar una aplicación en Java. Permiten representar situaciones específicas que no están

adecuadamente contempladas por las excepciones predefinidas.

Además, posibilitan establecer convenciones y patrones de manejo de errores dentro de la aplicación, mediante la creación de una jerarquía de excepciones que siga un diseño coherente.

Por otra parte, al definir excepciones personalizadas con mensajes descriptivos y claros, se facilita la comunicación de las causas de un error, tanto a quienes utilizan la aplicación como a quienes la desarrollan, lo que contribuye a una mejor comprensión y resolución del problema.

CONTINUAR

2. Buenas prácticas para manejar excepciones

A continuación, se presentan algunas recomendaciones para el manejo de excepciones en Java y en la mayoría de los lenguajes de programación:

- **Capturar las excepciones en el nivel adecuado**. Deben gestionarse en el punto donde realmente sea posible realizar una acción significativa para resolver el problema.

- **Incluir información relevante en los mensajes de excepción**: conviene incorporar datos que faciliten el diagnóstico, como detalles del contexto o de las operaciones que condujeron al error.

- **Utilizar `finally` cuando se trabajen recursos**: siempre que se empleen recursos que deban liberarse, como conexiones a bases de datos o

archivos, es recomendable usar bloques *finally* para garantizar su correcta liberación.

- **Evitar limitarse a imprimir un mensaje y continuar**: no resulta adecuado mostrar un error y seguir adelante sin una gestión apropiada. Las excepciones deben registrarse, informarse o manejarse correctamente.

- **Documentar las excepciones que puede lanzar un método**: es conveniente especificar tanto las excepciones predefinidas como las personalizadas. Esto facilita que quienes utilicen el código comprendan cómo deben manejarlas.

BENEFICIOS DEL MANEJO DE EXCEPCIONES EN JAVA

ERRORES

El manejo de excepciones proporciona diversos beneficios. A continuación, se presentan algunos de ellos:

- **Separación de la lógica de errores**. Permite separar la gestión de errores del código principal, lo que mejora la legibilidad y favorece una estructura más ordenada. El código destinado al manejo de errores se concentra en bloques *catch*, lo que facilita su identificación y comprensión.

- **Comunicación detallada de errores**: ofrece un mecanismo para comunicar información específica sobre el error, incluidos mensajes descriptivos, ubicaciones y causas. Esto favorece la identificación y resolución de problemas, tanto para quienes desarrollan como para quienes utilizan la aplicación.
- **Mejora de la escalabilidad y el mantenimiento**: una jerarquía de excepciones bien definida y documentada facilita la incorporación de nuevas excepciones para casos particulares y simplifica el mantenimiento del código existente.
- **Diseño más seguro y eficiente**: contribuye al desarrollo de aplicaciones seguras y eficientes, ya que permite una respuesta controlada ante situaciones excepcionales, evita posibles fugas de recursos y garantiza una adecuada liberación de los recursos utilizados.

BENEFICIOS DEL MANEJO DE EXCEPCIONES EN JAVA

ERRORES

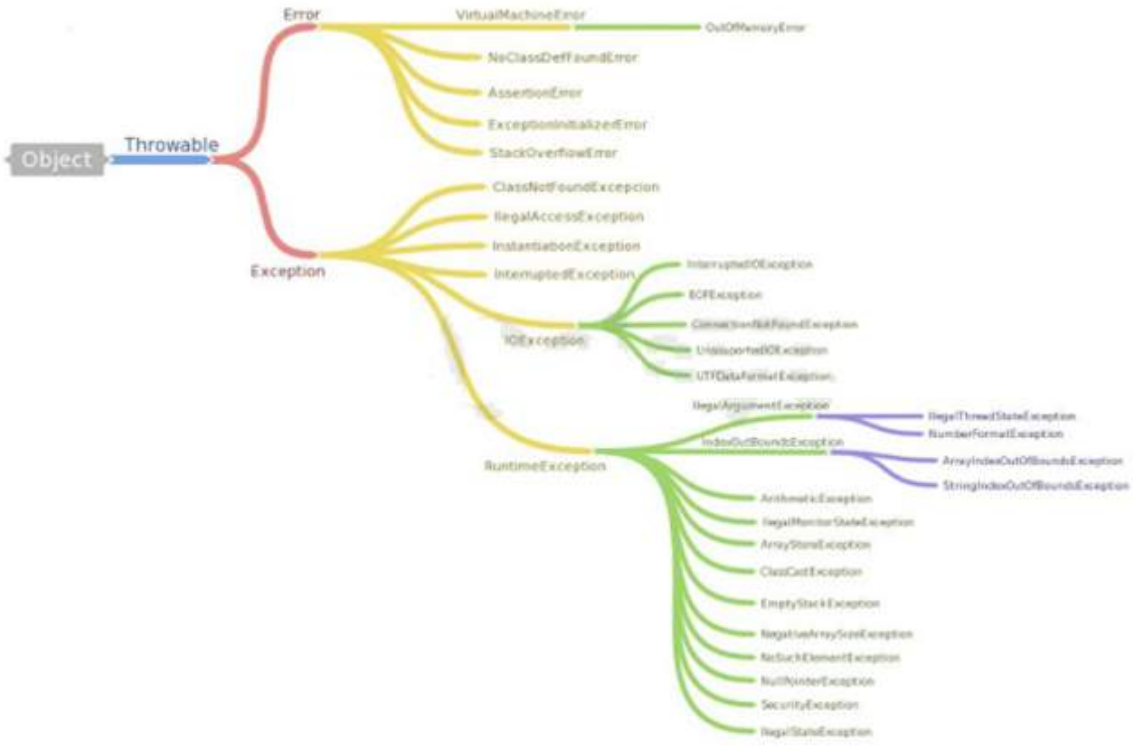
Además de las excepciones, en Java existen los errores, que son clases similares a las excepciones, pero cuyo propósito es informar sobre una situación anómala grave; es decir, situaciones que, en principio, no deberían ocurrir. Entre los ejemplos de errores se encuentran las siguientes clases:

- `ThreadDeath`
- `VirtualMachineError`
- `AssertionError`

Dado que los errores se consideran situaciones inesperadas, el compilador no exige que se capturen de manera obligatoria. Todos los errores deben extender de la clase Error.

El siguiente esquema muestra cómo se organizan las excepciones y los errores en Java, partiendo de la clase base Throwable y dividiéndose en Error y Exception, junto con algunas de sus clases más representativas.

Figura 5. Jerarquía de excepciones y errores en Java



Fuente: [imagen sin título sobre jerarquía de excepciones y errores en Java], (s.f.), https://goo.su/pd2X

CONTINUAR

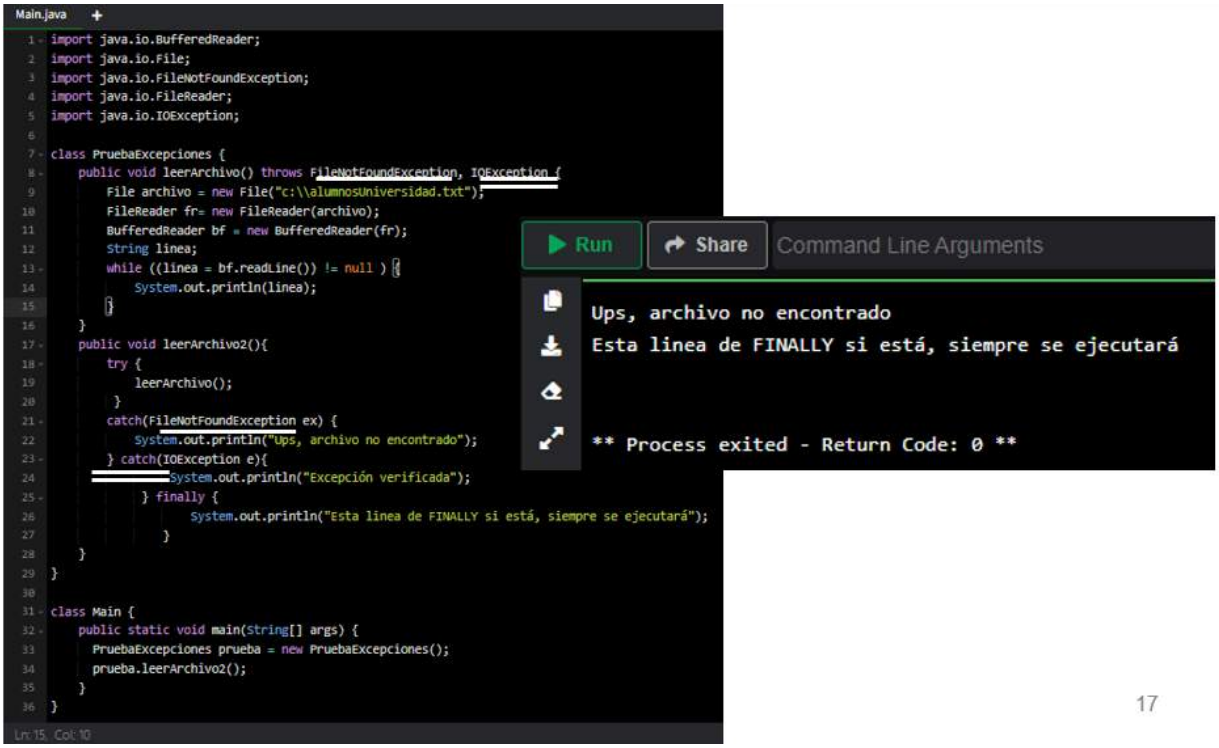
3. Throw

Para lanzar una excepción se debe utilizar la palabra reservada `throw`, seguida de un objeto del tipo `Exception` (o `Error`). Este recurso resulta útil cuando se necesita informar al programa invocador que se ha producido una situación anómala en el código.

```
public boolean debitarCuenta (int valor)

throws InvalidAmountException {

...

if (balance+sobregiro+sobrecanje < valor)

  throw new InvalidAmountException()

...

}
```

En la imagen que se presenta a continuación puede observarse que es posible considerar varias opciones, como se muestra en la línea de código 8, donde aparecen dos excepciones escritas después de la palabra reservada *throws*. Sin embargo, debe tenerse en cuenta que la misma cantidad de opciones declaradas debe definirse también en el manejo correspondiente, tal como se observa en las líneas 21 y 24 de esa misma imagen.

Figura 6. Declaración y manejo de múltiples excepciones con bloque *finally*



Fuente: captura de pantalla de Online-Java (www.online-java.com) https://www.online-java.com/2VKoJozdvg

---

Una vez finalizado el manejo de la excepción, el control del programa se reanuda después del último bloque `catch`. Las variables locales pueden perder su estado si el flujo de ejecución se interrumpe antes de completar su procesamiento. Es posible crear una excepción simplemente definiendo una clase que extienda de otra excepción existente:

```
public class InvalidAmountException extends RuntimeException {}
```

O bien:

```
public class InvalidAmountException extends Exception {}
```

Al tratarse de una clase, pueden añadirse los métodos y atributos que se consideren necesarios:

```
public class InvalidAmountException extends RuntimeException
```

```
{

    protected int valorInvalido;

    public InvalidAmountException(int valorInvalido) {

        this.valorInvalido = valorInvalido;

    }

    public int getValorInvalido() {

        return valorInvalido;
```

Excepciones comunes

Entre las excepciones más habituales se encuentran las siguientes:

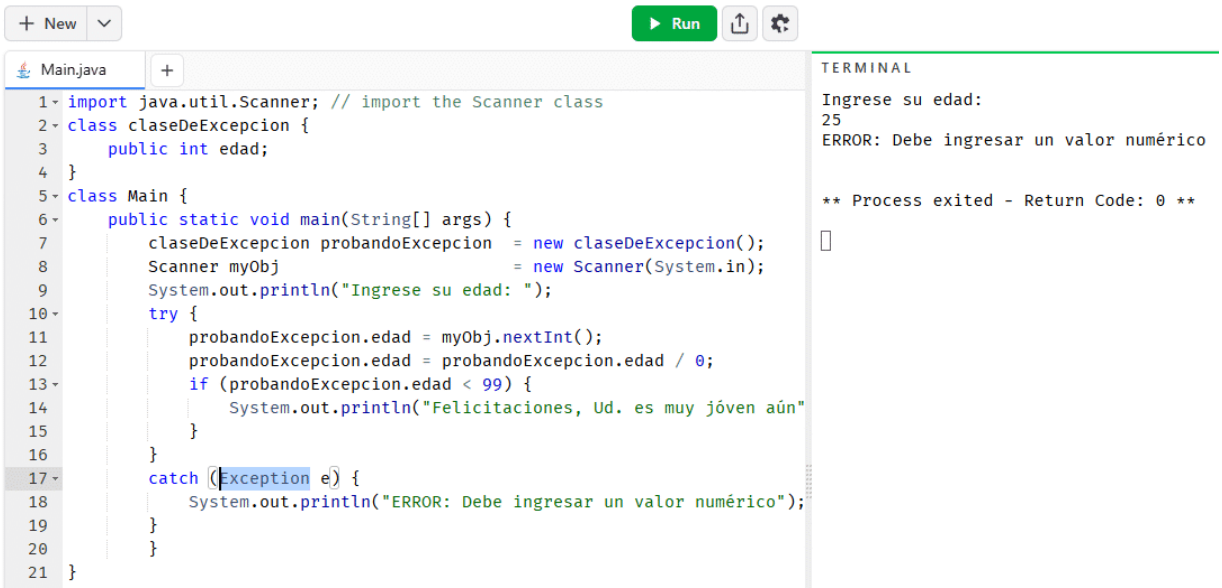
- **ArithmeticException**. Errores matemáticos, como la división por cero.
- **ArrayIndexOutOfBoundsException**: intento de acceder a una posición de índice no válida en un arreglo.

- **FileNotFoundException**: intento de acceder a un archivo que no existe.
- **IOException**: fallas de entrada o salida, como la imposibilidad de leer un archivo.

Capturando adecuadamente

Siempre es recomendable proporcionar un mensaje claro al usuario para indicarle el error. En los siguientes ejemplos se muestra la diferencia entre capturar una excepción de forma adecuada y no hacerlo correctamente.

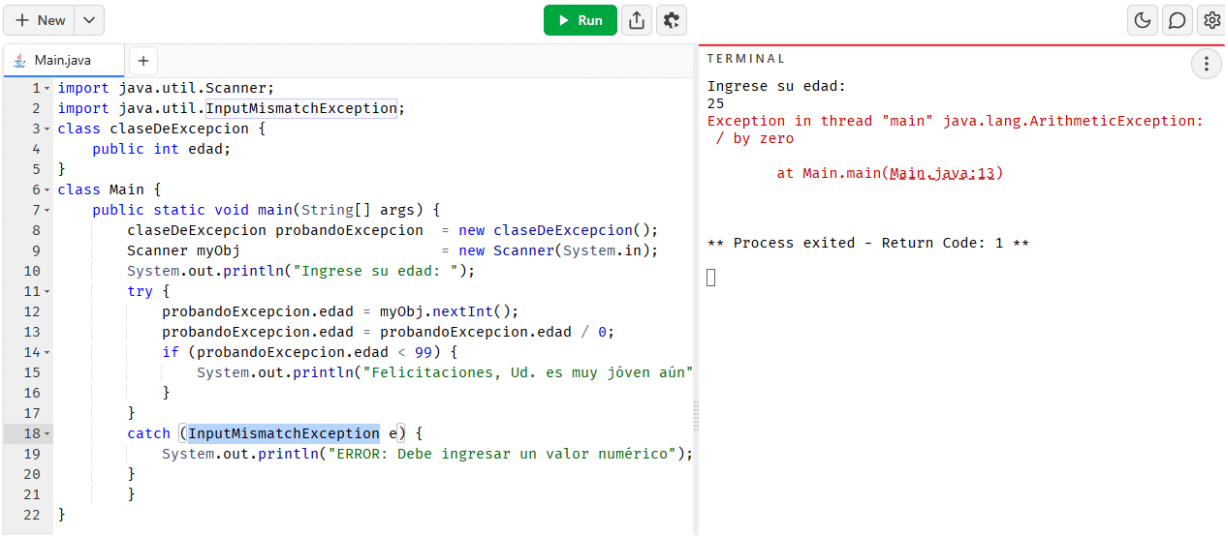
Figura 7. Captura adecuada de la excepción y mensaje informativo al usuario



Fuente: captura de pantalla de Online-Java (www.online-java.com)
https://www.online-java.com/3SjpdpvOWS

Teniendo en cuenta el mapa conceptual presentado anteriormente, puede advertirse que la captura no es la adecuada (véase la línea 18), por lo que el programa no logra interceptar la excepción y finaliza de forma abrupta.

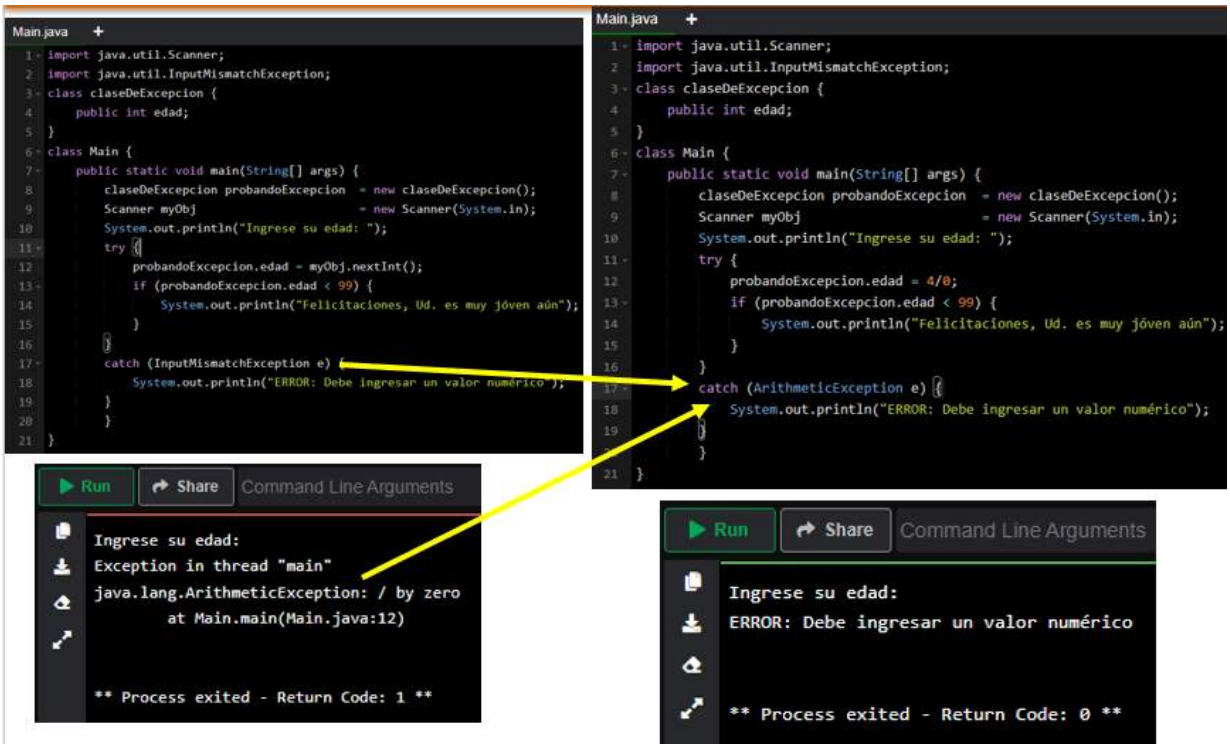
Figura 8. Captura incorrecta de la excepción y finalización abrupta del programa



Fuente: captura de pantalla de Online-Java (www.online-java.com
) https://www.online-java.com/3SjpdpvOWS

Si se ajusta correctamente el tipo de excepción que se desea capturar, puede observarse que siempre existe una forma más adecuada de interceptar un posible error e informar al usuario de manera clara y precisa, evitando que el programa finalice de forma abrupta.

Figura 9. Comparación entre captura incorrecta y captura adecuada de excepciones



Fuente: captura de pantalla de Online-Java (www.online-java.com)

CONTINUAR

Referencias

[Imagen sin título sobre jerarquía de excepciones y errores en Java], (s.f.). https://codejavu.blogspot.com/2013/06/manejo-de-excepciones-en-java.html#google_vignette

Referencias bibliográficas de consulta

Grupo C IS513. (s. f.). *Línea del tiempo de los lenguajes de programación*. https://www.timetoast.com/timelines/linea-del-tiempo-de-los-lenguajes-de-programacion-45ac36c7-d98e-4267-b5e3-d1d426cb469a

Joyanes Aguilar, L. (2009). *Fundamentos de programación. Algoritmos, estructura de datos y objetos*. McGraw-Hill.

López, G., Jeder, I. y Vega, A. (2009). *Análisis y diseño de algoritmos: Implementaciones en C y Pascal*. Alpha Editorial.

Pierro, B. (2018). El mundo mediado por algoritmos. *Revista Pesquisa*. https://revistapesquisa.fapesp.br/es/el-mundo-mediado-por-algoritmos/

CONTINUAR