



**UNIVERSIDAD NACIONAL DE INGENIERÍA**  
**FACULTAD DE CIENCIAS Y SISTEMAS**  
**DEPARTAMENTO DE INFORMÁTICA**

# Elementos fundamentales del Lenguaje de Programación (9.Excepciones)

Unidad II:

Docente: Ing. Danilo Noguera Rivera

 [danilo.noguera@gmail.com](mailto:danilo.noguera@gmail.com)

 [danilo.noguera@fcys.uni.edu.ni](mailto:danilo.noguera@fcys.uni.edu.ni)

 [.com/danilohnr](https://www.facebook.com/danilohnr)

 [.com/danilohnr](https://www.instagram.com/danilohnr)

 [.com@danilohnr](https://twitter.com/danilohnr)

 [danilo.noguera](https://www.whatsapp.com/danilo.noguera)

# Contenido

I. INTRODUCCIÓN A JAVA

II. VARIABLES

III. TIPOS DE DATOS

IV. OPERADORES Y EXPRESIONES

V. SENTENCIAS DE CONTROL

VI. ARREGLOS

VII. CADENAS

VIII. MÉTODOS

**IX. EXCEPCIONES**

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# ¿Qué son las excepciones?

- Es una condición anormal que surge en una secuencia de código en tiempo de ejecución (RUN TIME), generalmente esta secuencia de código está dentro de un método.
- Una excepción es un objeto que es lanzado cuando ocurre un error. Cuando ocurre una excepción, se crea un objeto de una clase de excepción particular.
- El tiempo de ejecución solo puede detectar errores dentro de métodos. Puede ser un método constructor o un método general o un método estático.

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Exception.html>

# IX. EXCEPCIONES

---

## INTRODUCCIÓN A LAS EXCEPCIONES

¿Qué son las excepciones?

- Las excepciones son como eventos que ocurren cuando algo sale mal en el programa.
- Son como un tipo de error.
- Las excepciones no tienen que significar que el programa se detendrá completamente, sino que pueden ser manejados con un protocolo específico.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

¿Para qué se  
usan las  
excepciones?

- Java usa las excepciones para representar problemas y para proveer control de ejecución limpio y fácil de leer cuando esos problemas surgen.
- Normalmente el desarrollador primero programa pensando en que el flujo de ejecución de sus programas siempre va a funcionar exitosamente.
- Después escriben el código para lidiar cuando algo malo sucede con el programa.
- Es un enfoque complicado ya que la mayoría de errores surgen cuando el programa es utilizado por el usuario final.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

¿Para qué se  
usan las  
excepciones?

- Generalmente el desarrollador informa que el programa está 90% completado, pero no especifica que está 90% completado para los casos en que el programa funcione correctamente.
- Por eso es que las aplicaciones deben ser sometidas a pruebas para que surjan todos los problemas posibles y así buscar cómo solucionarlos.
- Java permite realizar algunas acciones que podrían lidiar con una gran variedad de problemas.
- Generalmente Java trata de categorizar los problemas.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- Las excepciones pueden ser divididas en las siguientes 2 maneras, ya que representan categorías significativas de problemas:
  1. Pueden haber problemas en el programa en sí. Cosas que hace incorrectamente. Por ejemplo, tratar de acceder a elementos que no existen en un arreglo.
  2. Y están las causas externas al programa, por ejemplo, el mal uso de los usuarios. Por ejemplo, que el usuario ingrese datos de un archivo que no existe o ingrese datos de tipo incorrecto. Otro ejemplo podría ser que el usuario está haciendo uso de recursos de red y el medio de conexión falla de repente. Son problemas del entorno. El buen código trata de corregir el error, reconociendo estos problemas externos y haciéndole saber al usuario cómo corregirlos.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- En la práctica, Java usa 3 categorías amplias de problemas:
  1. Las excepciones que representan problemas con el entorno, caen en la categoría de **excepciones revisadas/verificadas**. En este caso, el compilador insiste en que se debe realizar un esfuerzo para identificar el problema y escribir código que intente resolverlo.
  2. Luego, está la categoría que comprende los tipos de problemas que el desarrollador piensa que no deberían suceder, y por tanto, no se intentan resolver. Solo se revisa (debug) el código. Se les llama **excepciones en tiempo de ejecución (runtime)**. Esta categoría cae en una categoría más amplia llamada **excepciones no revisadas/no verificadas**.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- En la práctica, Java usa 3 categorías amplias de problemas:
  3. La tercer categoría se llama **errores**. Al igual que la primera categoría, representa problemas externos al programa. Sin embargo, Java no requiere que hagamos algo al respecto para tratar de corregir errores, debido a son considerados catastróficos e irrecuperables. Por ejemplo, si el programa se queda sin memoria, no hay mucho que un desarrollador pueda hacer al respecto. Esta categoría también forma parte de la categoría de **excepciones no revisadas/no verificadas**

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

**Excepciones verificadas**

**Throwable**



**Throwable** es la clase raíz de la excepciones en Java. Tiene 2 subclases:

- **Error**, el cual trae otros errores individuales.
- **Exception**, el cual trae otras subclases. Incluyendo la subclase **RuntimeException**.

**Excepciones no verificadas**

**Excepciones en tiempo de ejecución**

**RuntimeException**

**Errores**

**Error**

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Module java.base  
Package java.lang

### Class Exception

java.lang.Object  
  java.lang.Throwable  
    java.lang.Exception

All Implemented Interfaces:  
Serializable

#### Direct Known Subclasses:

AbsentInformationException, AclNotFoundException, ActivationException, AgentInitializationException, AgentLoadException, AlreadyBoundException, AttachNotSupportedException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CardException, CertificateException, ClassNotLoadedException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionControl.ExecutionControlException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSEException, IllegalClassFormatException, IllegalConnectorArgumentsException, IncompatibleThreadStateException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvalidTypeException, InvocationException, IOException, JMXException, JShellException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SQLException, StringConcatException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, VMStartException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

La súper clase de  
Excepciones

Las clases de excepciones que heredan de  
RuntimeException son conocidas como  
Excepciones no verificadas

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Exception.html>

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Module java.base  
Package java.lang

**Class RuntimeException**

java.lang.Object  
  java.lang.Throwable  
    java.lang.Exception  
      java.lang.RuntimeException

All Implemented Interfaces:  
Serializable

Direct Known Subclasses:

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, CatalogException, ClassCastException, ClassNotPreparedException, CMMException, CompletionException, ConcurrentModificationException, DateTimeException, DOMException, DuplicateRequestException, EmptyStackException, EnumConstantNotPresentException, EventException, FileSystemAlreadyExistsException, FileSystemNotFoundException, FileNotFoundException, IllegalArgumentException, IllegalCallerException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, IllformedLocaleException, ImagingOpException, InaccessibleObjectException, IncompleteAnnotationException, InconsistentDebugInfoException, IndexOutOfBoundsException, InternalException, InvalidCodeIndexException, InvalidLineNumberException, InvalidModuleDescriptorException, InvalidModuleException, InvalidRequestStateException, InvalidStackFrameException, JarSignerException, JMRuntimeException, JSEException, LayerInstantiationException, LSEException, MalformedParameterizedTypeException, MalformedParametersException, MirroredTypesException, MissingResourceException, NashornException, NativeMethodException, NegativeArraySizeException, NoSuchDynamicMethodException, NoSuchElementException, NoSuchMechanismException, NullPointerException, ObjectCollectedException, ProfileDataException, ProviderException, ProviderNotFoundException, RangeException, RasterFormatException, RejectedExecutionException, ResolutionException, SecurityException, SPIResolutionException, TypeNotPresentException, UncheckedIOException, UndeclaredThrowableException, UnknownEntityException, UnknownTreeException, UnmodifiableModuleException, UnmodifiableSetException, UnsupportedOperationException, VMDisconnectedException, VMMismatchException, VMOutOfMemoryException, WrongMethodTypeException, XPathException

La clase de Excepciones en tiempo de ejecución (RuntimeException)

Las subclases de excepciones en tiempo de ejecución (Runtime).

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/RuntimeException.html>

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Module java.base  
Package java.lang

### Class Exception

java.lang.Object  
  java.lang.Throwable  
    java.lang.Exception

All Implemented Interfaces:  
Serializable

#### Direct Known Subclasses:

AbsentInformationException, AclNotFoundException, ActivationException, AgentInitializationException, AgentLoadException, AlreadyBoundException, AttachNotSupportedException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CardException, CertificateException, ClassNotLoadedException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionControl.ExecutionControlException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSEException, IllegalClassFormatException, IllegalConnectorArgumentsException, IncompatibleThreadStateException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvalidTypeException, InvocationException, IOException, JMEException, JShellException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SQLException, StringConcatException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, VMStartException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

La súper clase de  
Excepciones

Todas las clases de excepciones que  
heredan directamente de Exception son  
conocidas como Excepciones verificadas

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Exception.html>

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Enfoques del manejo de excepciones

- Reanudación, se espera manejar el error de manera que la situación que causó el error pueda ser corregida, y luego el código que tenía fallas, pueda ser re-ejecutado.
- Terminación, en este caso los errores son serios y es imposible permitir que el programa siga ejecutándose exitosamente.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Las palabras  
reservadas para  
el manejo de  
excepciones

- `try`
- `catch`
- `finally`



No se puede tener un bloque `try` sin al menos un bloque `catch`. El bloque `finally` es opcional.

- `throw`
- `throws`



Se usan para crear nuestras propias excepciones.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# El bloque `try...catch`

- El bloque `try...catch` se usa para determinar cómo las excepciones alteran el flujo normal de los programas.
- Ahora se escribirá código para manejar las situaciones que surgen cuando ocurren las excepciones.
- Esto es posible gracias al mecanismo `try...catch`.



# IX. EXCEPCIONES

---

## INTRODUCCIÓN A LAS EXCEPCIONES

El bloque  
`try...catch`

- Al manejar excepciones, tenemos 2 elementos importantes a tomar en cuenta. Es de mucha utilidad separar lo que pasa con el código cuando las cosas pasan según lo planeado, del código cuando las cosas no suceden bien.

## IX. EXCEPCIONES

# INTRODUCCIÓN A LAS EXCEPCIONES



# Vs.



if.....



if.....



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

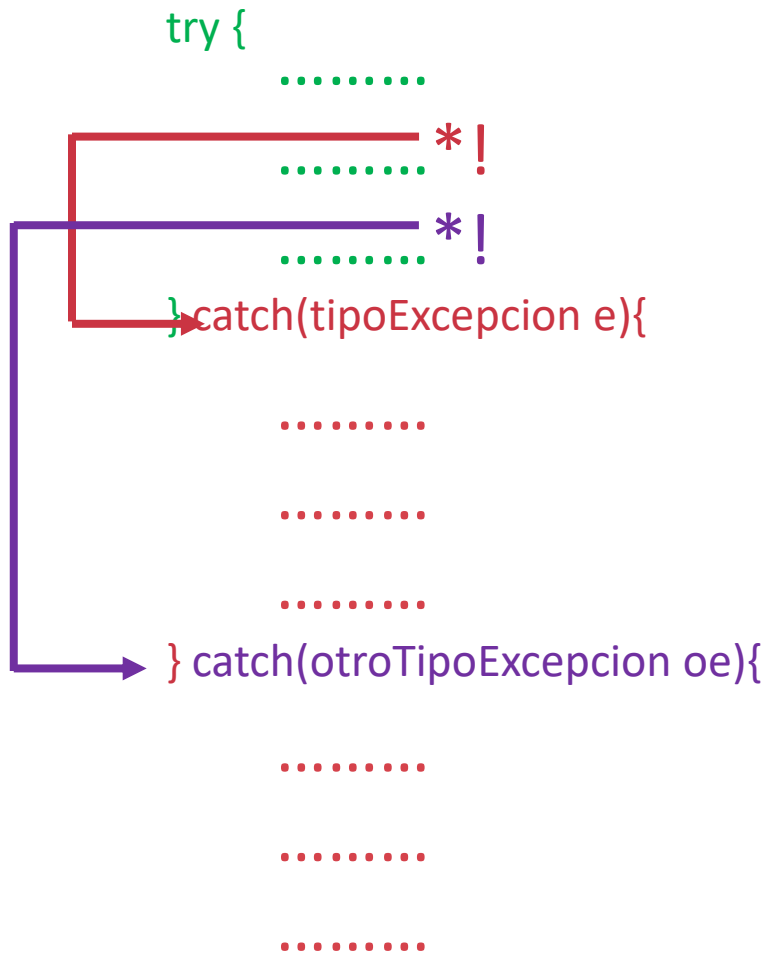
```
try {  
    .....  
    .....  
    .....  
} catch(tipoExcepcion e){  
    .....  
    .....  
    .....  
} catch(otroTipoExcepcion oe){  
    .....  
    .....  
    .....  
}
```



- El bloque **try** le dice al sistema que si todo está bien, que ejecute todo este código en el bloque.
- El bloque **catch** va con el nombre del tipo de excepción que describa la categoría de problema que podría surgir, luego un nombre de variable local que guardará el valor de la excepción lanzada. En este bloque va a el código para la recuperación ante el problema.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES



- Si algo lanza o genera una excepción, la ejecución saltará directamente de ahí al primer bloque `catch`. El objeto pasado como `tipoExcepcion` se convierte en un parámetro formal para ser usado opcionalmente dentro del bloque.
- Es muy probable que algo más salga mal en cualquier parte del código, así que es normal tener más bloques `catch` que capturen otro tipo de excepción.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
ExcepcionesNoVerificadas.java ✕
1 package excepciones;
2
3 public class ExcepcionesNoVerificadas {
4
5     public static void main(String[] args) {
6         System.out.println("=====INICIO DEL PROGRAMA=====");
7         int a = 12, b = 0, respuesta = 0;
8         respuesta = a/b;
9         System.out.println("a = " + a + " b = " + b + " respuesta = " + respuesta);
10
11         double [] valores = new double[3];
12         valores[0] = 44.77;
13         valores[1] = 99;
14         valores[2] = 12.6;
15         valores[3] = 100.963;
16
17         for (double d : valores) {
18             System.out.println("d = " + d);
19         }
20     }
21 }
```

```
Exception in thread "main" =====INICIO DEL PROGRAMA=====
java.lang.ArithmeticException: / by zero
    at excepciones.ExcepcionesNoVerificadas.main(ExcepcionesNoVerificadas.java:8)
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
ExcepcionesNoVerificadas.java ✕
1 package excepciones;
2
3 public class ExcepcionesNoVerificadas {
4
5     public static void main(String[] args) {
6         System.out.println("=====INICIO DEL PROGRAMA=====");
7         int a = 12, b = 6, respuesta = 0;
8         respuesta = a/b;
9         System.out.println("a = " + a + " b = " + b + " respuesta = " + respuesta);
10
11         double [] valores = new double[3];
12         valores[0] = 44.77;
13         valores[1] = 99;
14         valores[2] = 12.6;
15         valores[3] = 100.963;
16
17         for (double d : valores) {
18             System.out.println("d = " + d);
19         }
20     }
21 }
```

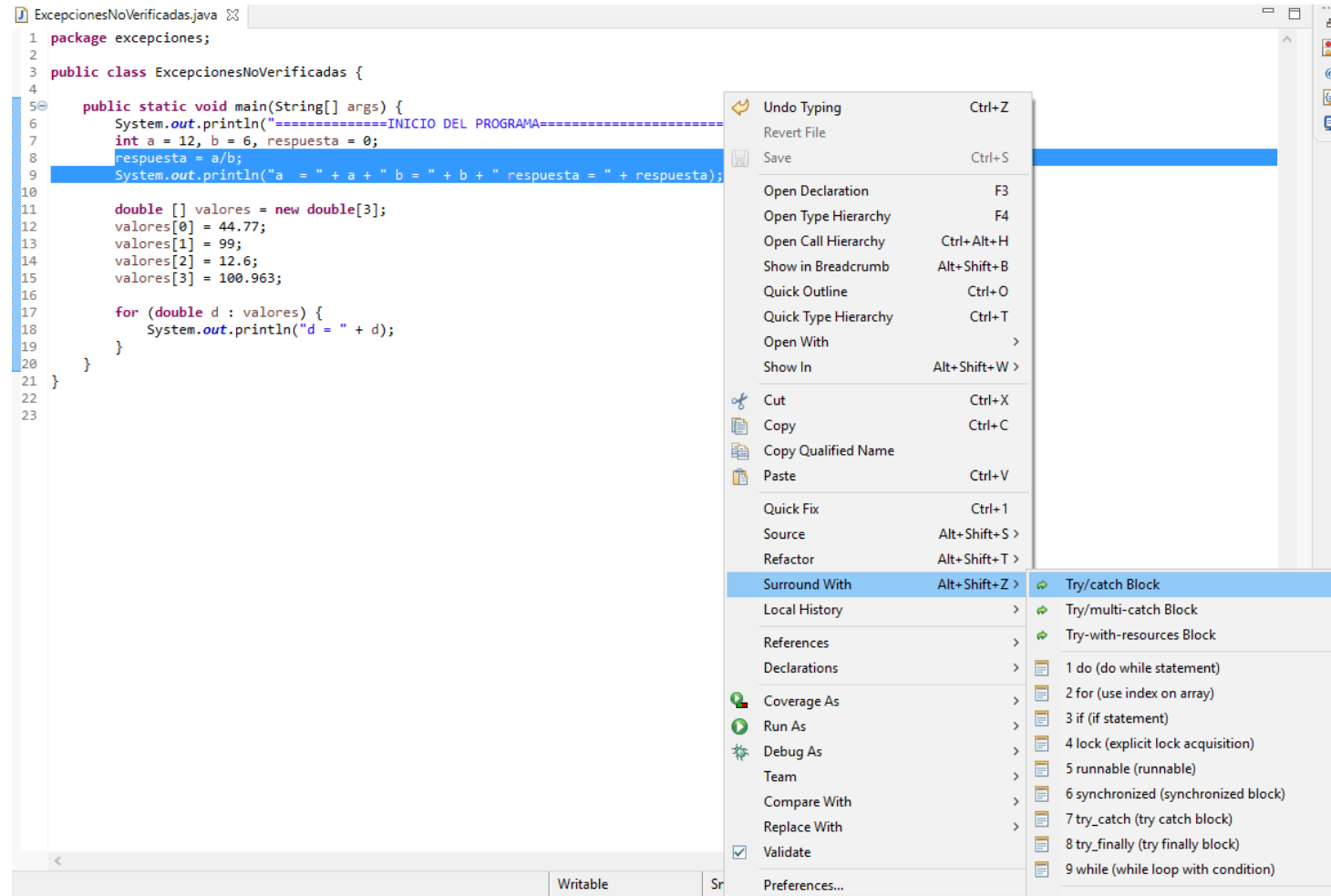
=====INICIO DEL PROGRAMA=====

a = 12 b = 6 respuesta = 2

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException](#): Index 3 out of bounds for length 3  
at excepciones.ExcepcionesNoVerificadas.main([ExcepcionesNoVerificadas.java:15](#))

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

ExcepcionesNoVerificadas.java

```
1 package excepciones;
2
3 public class ExcepcionesNoVerificadas {
4
5     public static void main(String[] args) {
6         System.out.println("=====INICIO DEL PROGRAMA=====");
7         int a = 12, b = 0, respuesta = 0;
8         try {
9             respuesta = a/b;
10            System.out.println("a = " + a + " b = " + b + " respuesta = " + respuesta);
11        } catch (ArithmeticException e) {
12            System.out.println("Ocurrió un ERROR aritmético!");
13            e.printStackTrace();
14        }
15        System.out.println("Sentencia después del bloque try...catch");
16
17        double [] valores = new double[3];
18        valores[0] = 44.77;
19        valores[1] = 99;
20        valores[2] = 12.6;
21        //valores[3] = 100.963;
22
23        for (double d : valores) {
24            System.out.println("d = " + d);
25        }
26    }
27 }
```

=====INICIO DEL PROGRAMA=====

Ocurrió un ERROR aritmético!

[java.lang.ArithmeticException](#): / by zero

at excepciones.ExcepcionesNoVerificadas.main([ExcepcionesNoVerificadas.java:9](#))

Sentencia después del bloque try...catch

d = 44.77

d = 99.0

d = 12.6



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
ExcepcionesNoVerificadas.java
1 package excepciones;
2
3 public class ExcepcionesNoVerificadas {
4
5     public static void main(String[] args) {
6         System.out.println("=====INICIO DEL PROGRAMA=====");
7         int a = 12, b = 0, respuesta = 0;
8         try {
9             respuesta = a/b;
10            System.out.println("a = " + a + " b = " + b + " respuesta = " + respuesta);
11        } catch (ArithmeticException e) {
12            System.out.println("Ocurrió un ERROR aritmético!");
13            //e.printStackTrace();
14        }
15        System.out.println("Sentencia después del bloque try...catch");
16
17        double [] valores = new double[3];
18        valores[0] = 44.77;
19        valores[1] = 99;
20        valores[2] = 12.6;
21        //valores[3] = 100.963;
22
23        for (double d : valores) {
24            System.out.println("d = " + d);
25        }
26    }
27 }
```

```
=====INICIO DEL PROGRAMA=====
Ocurrió un ERROR aritmético!
Sentencia después del bloque try...catch
d = 44.77
d = 99.0
d = 12.6
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

¿En qué  
orden ubicar  
los bloques  
catch?

- Un bloque **catch** puede atrapar todas las excepciones de un tipo específico o bien todos los tipos de excepciones.
- Si en el primer bloque **catch** se declara una excepción empleando la clase **Exception**, entonces ese bloque **catch** puede atrapar todos los tipos de excepciones, ya que la clase **Exception** es la superclase de todas las clases de excepciones.
- Los bloques **catch** restantes asociados con ese bloque **try** se ignoran.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

¿En qué  
orden ubicar  
los bloques  
catch?

- Se debe tener cuidado acerca del orden en el cual se listan los bloques `catch` que siguen a un bloque `try`.
- Esto se logra poniendo primero excepciones más específicas, y luego excepciones más generales.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

EjemploExcepcion.java

```
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class EjemploExcepcion {
7     static Scanner console = new Scanner(System.in);
8     public static void main(String[] args) {
9         int dividendo, divisor, cociente;
10        try {
11            System.out.print("Ingrese el dividendo: ");
12            dividendo = console.nextInt();
13            System.out.println();
14            System.out.print("Ingrese el divisor: ");
15            divisor = console.nextInt();
16            System.out.println();
17            cociente = dividendo / divisor;
18            System.out.println("Cociente = " + cociente);
19        } catch (ArithmeticException e) {
20            System.out.println(e.toString());
21        } catch (InputMismatchException e) {
22            System.out.println(e.toString());
23        }
24    }
25 }
```

Ingrese el dividendo: 5

Ingrese el divisor: 0

[java.lang.ArithmeticException](#): / by zero

---

Ingrese el dividendo: 5e

[java.util.InputMismatchException](#)

---

Ingrese el dividendo: 15

Ingrese el divisor: h

[java.util.InputMismatchException](#)

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
EjemploExcepcion.java
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class EjemploExcepcion {
7     static Scanner console = new Scanner(System.in);
8     public static void main(String[] args) {
9         int dividendo, divisor, cociente;
10        try {
11            System.out.print("Ingrese el dividendo: ");
12            dividendo = console.nextInt();
13            System.out.println();
14            System.out.print("Ingrese el divisor: ");
15            divisor = console.nextInt();
16            System.out.println();
17            cociente = dividendo / divisor;
18            System.out.println("Cociente = " + cociente);
19        } catch (ArithmeticException e) {
20            System.out.println(e.toString());
21        } catch (InputMismatchException e) {
22            String str;
23            str = console.next();
24            System.out.println(e.toString() + " " + str);
25        }
26    }
27 }
```

Ingrese el dividendo: 5

Ingrese el divisor: 0

[java.lang.ArithmeticException](#): / by zero

---

Ingrese el dividendo: 59

Ingrese el divisor: 2h2

[java.util.InputMismatchException](#) 2h2

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
EjemploExcepcion.java ✖
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class EjemploExcepcion {
7     static Scanner console = new Scanner(System.in);
8     public static void main(String[] args) {
9         int dividendo, divisor, cociente;
10        try {
11            System.out.print("Ingrese el dividendo: ");
12            dividendo = console.nextInt();
13            System.out.println();
14            System.out.print("Ingrese el divisor: ");
15            divisor = console.nextInt();
16            System.out.println();
17            cociente = dividendo / divisor;
18            System.out.println("Cociente = " + cociente);
19        } catch (Exception e) {
20            System.out.println(e.toString());
21        } catch (InputMismatchException e) {
22            System.out.println(e.toString());
23        }
24    }
25 }
```

```
EjemploExcepcion.java ✖
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class EjemploExcepcion {
7     static Scanner console = new Scanner(System.in);
8     public static void main(String[] args) {
9         int dividendo, divisor, cociente;
10        try {
11            System.out.print("Ingrese el dividendo: ");
12            dividendo = console.nextInt();
13            System.out.println();
14            System.out.print("Ingrese el divisor: ");
15            divisor = console.nextInt();
16            System.out.println();
17            cociente = dividendo / divisor;
18            System.out.println("Cociente = " + cociente);
19        } catch (Exception e) {
20            System.out.println(e.toString());
21        } catch (InputMismatchException e) {
22            System.out.println(e.toString());
23        }
24    }
25 }
```

Unreachable catch block for InputMismatchException. It is already handled by the catch block for Exception

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
EjemploExcepcion.java ✕
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class EjemploExcepcion {
7     static Scanner console = new Scanner(System.in);
8     public static void main(String[] args) {
9         int dividendo, divisor, cociente;
10        try {
11            System.out.print("Ingrese el dividendo: ");
12            dividendo = console.nextInt();
13            System.out.println();
14            System.out.print("Ingrese el divisor: ");
15            divisor = console.nextInt();
16            System.out.println();
17            cociente = dividendo / divisor;
18            System.out.println("Cociente = " + cociente);
19        } catch (Exception e) {
20
21            if (e instanceof ArithmeticException) {
22                System.out.println(e.toString());
23            } else if (e instanceof InputMismatchException) {
24                System.out.println(e.toString());
25            }
26
27        }
28    }
29 }
```

Ingrese el dividendo: 10

Ingrese el divisor: 0

[java.lang.ArithmeticException](#): / by zero

---

Ingrese el dividendo: 10

Ingrese el divisor: v

[java.util.InputMismatchException](#)

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

- Hay varios tipos de excepciones, como excepciones de E/S, de falta de coincidencia de entrada, del formato de números, de archivo no encontrado y excepciones de índice de arreglo fuera de límite.
- Las clases para tratar excepciones de E/S, como la de archivo no encontrado, están contenidas en el paquete `java.io`.
- De igual forma, las clases para tratar excepciones de formato de número y aritméticas, como la división entre cero, están contenidas en el paquete `java.lang`.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

```
public Throwable  
    //Constructor predeterminado  
    //Crea una instancia de Throwable con una cadena de mensaje vacío
```

```
public Throwable(String strMessage)  
    //Constructor con parametros  
    //Crea una instancia de Throwable con una cadena de mensaje  
    //especificada por el parametro strMessage
```

```
public String getMessage()  
    //Retorna el mensaje detallado almacenado en el objeto
```

```
public void printStackTrace()  
    //Metodo para imprimir el volcado de pila mostrando la secuencia de  
    //llamadas al metodo cuando ocurre una excepcion
```

```
public void printStackTrace(PrintWriter stream)  
    //Metodo para imprimir el volcado de pila mostrando la secuencia de  
    //llamadas al metodo cuando ocurre una excepcion. La salida se envia  
    //al flujo especificado por el flujo de parametros.
```

```
public String toString()  
    //Retorna una representacion en cadena del objeto Throwable
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

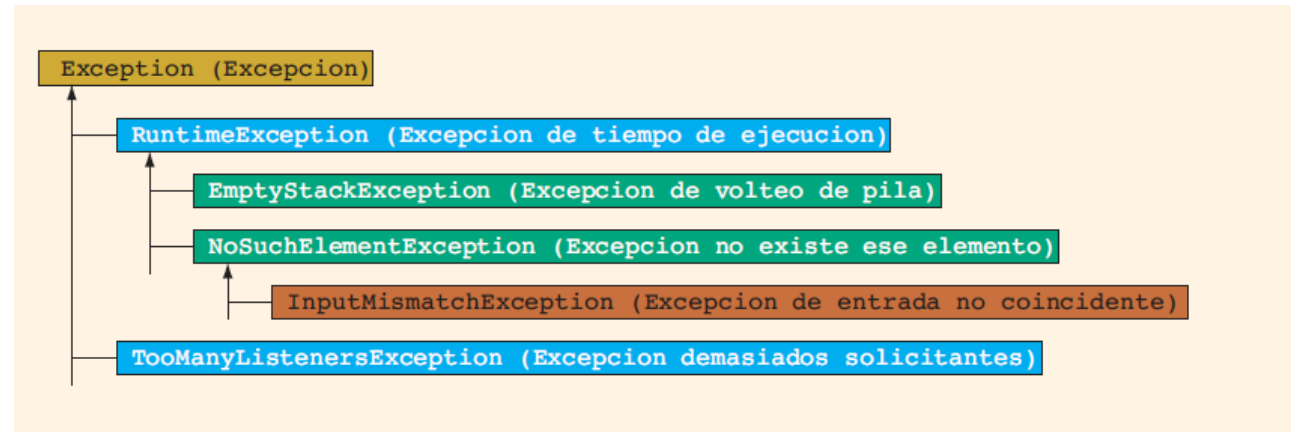
# Algunas Clases de excepciones comunes y sus métodos



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

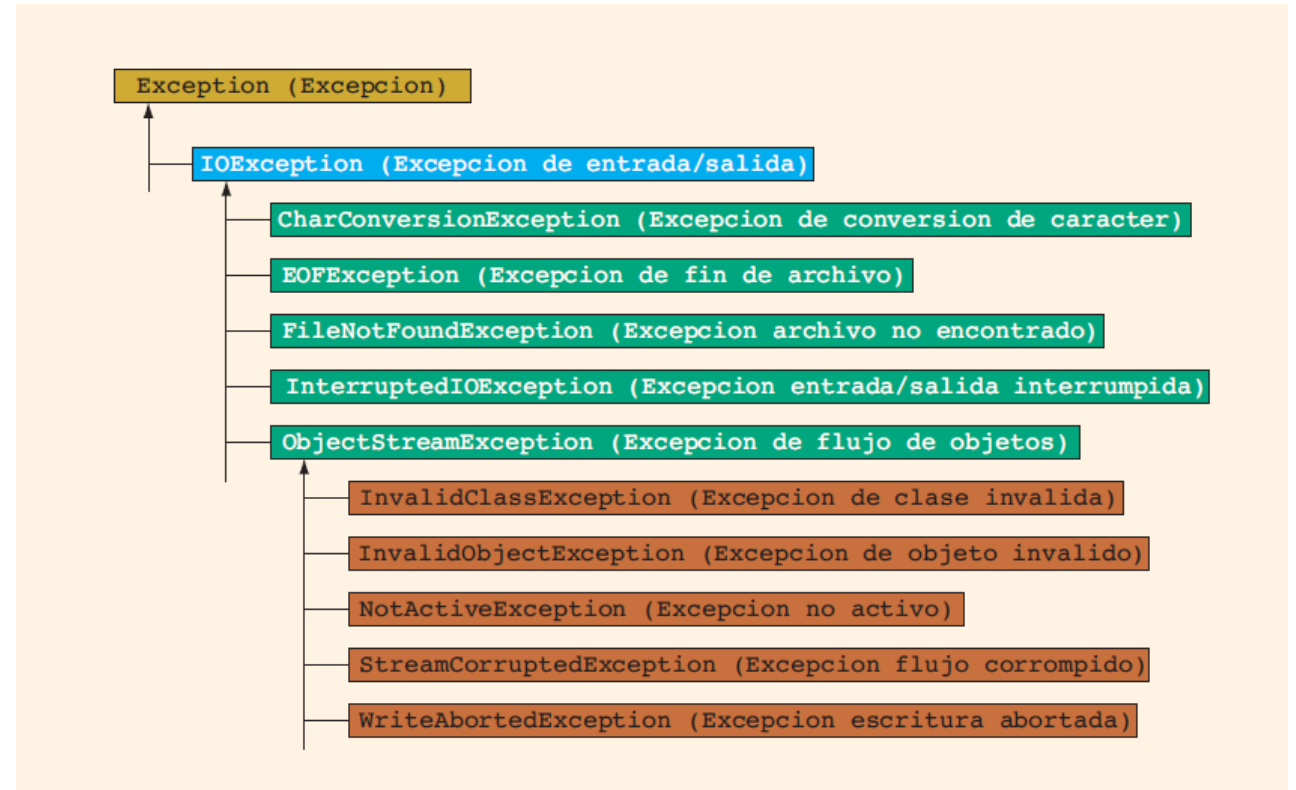
# Algunas Clases de excepciones comunes y sus métodos



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

Clase de excepción	Descripción
<code>ArithmeticException</code>	Errores aritméticos como división entre cero
<code>ArrayIndexOutOfBoundsException</code>	El índice del arreglo es menor que 0 o mayor que o igual a la longitud del arreglo.
<code>FileNotFoundException</code>	Referencia a un archivo que no se puede encontrar
<code>IllegalArgumentException</code>	Invocación a un método con argumentos ilegales
<code>IndexOutOfBoundsException</code>	Un arreglo o un índice de una cadena está fuera de límites.
<code>NullPointerException</code>	Referencia a un objeto que no se ha convertido en instancia
<code>NumberFormatException</code>	Uso de un formato ilegal de número
<code>StringIndexOutOfBoundsException</code>	Un índice de cadena es menor que 0 o mayor que o igual a la longitud de la cadena.
<code>InputMismatchException</code>	La entrada (símbolo) recuperada no coincide con el patrón para el tipo esperado o el símbolo está fuera del alcance para el tipo esperado.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

- Excepciones lanzadas por el método **nextInt()**

Clase de excepción	Descripción
<code>InputMismatchException</code>	Si la siguiente entrada (símbolo) no es un entero o está fuera de alcance
<code>NoSuchElementException</code>	Si la entrada está agotada
<code>IllegalStateException</code>	Si este explorador está cerrado

- Excepciones lanzadas por el método **nextDouble()**

Clase de excepción	Descripción
<code>InputMismatchException</code>	Si la siguiente entrada (símbolo) no es un número de punto flotante o está fuera de alcance
<code>NoSuchElementException</code>	Si la entrada está agotada
<code>IllegalStateException</code>	Si este explorador está cerrado

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

- Excepciones lanzadas por el método **next()**

Clase de excepción	Descripción
<code>NoSuchElementException</code>	Si ya no hay entrada (símbolos)
<code>IllegalStateException</code>	Si este explorador está cerrado

- Excepciones lanzadas por el método **nextLine()**

Clase de excepción	Descripción
<code>NoSuchElementException</code>	Si la entrada está agotada
<code>IllegalStateException</code>	Si este explorador está cerrado

- Excepciones lanzadas por el método **hasNext()**

Clase de excepción	Descripción
<code>IllegalStateException</code>	Si este explorador está cerrado

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Algunas Clases de excepciones comunes y sus métodos

- Excepciones lanzadas por métodos de **Integer**

Método	Excepción lanzada	Descripción
<code>parseInt(String str)</code>	<code>NumberFormatException</code>	La cadena <code>str</code> no contiene un valor <code>int</code> .
<code>valueOf(String str)</code>	<code>NumberFormatException</code>	La cadena <code>str</code> no contiene un valor <code>int</code> .

- Excepciones lanzadas por métodos de **Double**

Método	Excepción lanzada	Descripción
<code>parseDouble(String str)</code>	<code>NumberFormatException</code>	La cadena <code>str</code> no contiene un valor <code>double</code> .
<code>valueOf(String str)</code>	<code>NumberFormatException</code>	La cadena <code>str</code> no contiene un valor <code>double</code> .



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- Excepciones lanzadas por métodos de **String**

# Algunas Clases de excepciones comunes y sus métodos

Método	Excepción lanzada	Descripción
<code>String(String str)</code>	<code>NullPointerException</code>	<code>str</code> es <code>null</code> .
<code>charAt(int a)</code>	<code>StringIndexOutOfBoundsException</code>	El valor de <code>a</code> no es un índice válido.
<code>indexOf(String str)</code>	<code>NullPointerException</code>	<code>str</code> es <code>null</code> .
<code>lastIndexOf(String str)</code>	<code>NullPointerException</code>	<code>str</code> es <code>null</code> .
<code>substring(int a)</code>	<code>StringIndexOutOfBoundsException</code>	El valor de <code>a</code> no es un índice válido.
<code>substring(int a, int b)</code>	<code>StringIndexOutOfBoundsException</code>	El valor de <code>a</code> y/o <code>b</code> no es un índice válido.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Module java.base

Package java.io

### Class FileReader

java.lang.Object  
  java.io.Reader  
    java.io.InputStreamReader  
      java.io.FileReader

#### Constructors

##### Constructor

`FileReader(File file)`

#### FileReader

```
public FileReader(File file)  
    throws FileNotFoundException
```

Creates a new `FileReader`, given the `File` to read, using the platform's default charset.

##### Parameters:

file - the `File` to read

##### Throws:

`FileNotFoundException` - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

El método `FileReader()` lanza una excepción explícita o verificada, por lo tanto, hay que atraparla si surge. En tiempo de ejecución, Java no se va a encargar de verificarlo por nosotros.

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/FileReader.html>

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

ExcepcionesVerificadas.java x nombres.txt

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
```

ExcepcionesVerificadas.java x nombres.txt

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        Unhandled exception type FileNotFoundException[archivo];
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
ExcepcionesVerificadas.java  nombres.txt
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
```

```
ExcepcionesVerificadas.java  nombres.txt
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        Unhandled exception type IOException lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

ExcepcionesVerificadas.java x nombres.txt

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
```

ExcepcionesVerificadas.java x nombres.txt

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            lectorBuffer.readLine();
22        }
23    }
24 }
```

Unhandled exception type IOException

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

The screenshot shows an IDE window with the file 'ExcepcionesVerificadas.java' open. The code is as follows:

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        lectorArchivo = new FileReader(archivo);
16        lectorBuffer = new BufferedReader(lectorArchivo);
17
18        nombreEmpresa = lectorBuffer.readLine();
19        while(nombreEmpresa != null) {
20            System.out.println("Nombre de la empresa: " + nombreEmpresa);
21            nombreEmpresa = lectorBuffer.readLine();
22        }
23    }
24 }
25
26
```

A context menu is open over the code block from line 14 to 22. The menu items are:

- Undo Typing (Ctrl+Z)
- Revert File
- Save (Ctrl+S)
- Open Declaration (F3)
- Open Type Hierarchy (F4)
- Open Call Hierarchy (Ctrl+Alt+H)
- Show in Breadcrumb (Alt+Shift+B)
- Quick Outline (Ctrl+O)
- Quick Type Hierarchy (Ctrl+T)
- Open With >
- Show In (Alt+Shift+W >)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Copy Qualified Name
- Paste (Ctrl+V)
- Quick Fix (Ctrl+I)
- Source (Alt+Shift+S >)
- Refactor (Alt+Shift+T >)
- Surround With (Alt+Shift+Z >)
- Local History >
- References >
- Declarations >
- Coverage As >
- Run As >
- Debug As >
- Team >
- Compare With >
- Replace With >
- Validate
- Preferences...

The 'Surround With' submenu is open, showing the following options:

- Try/catch Block
- Try/multi-catch Block
- Try-with-resources Block
- 1 do (do while statement)
- 2 for (use index on array)
- 3 if (if statement)
- 4 lock (explicit lock acquisition)
- 5 runnable (runnable)
- 6 synchronized (synchronized block)
- 7 try\_catch (try catch block)
- 8 try\_finally (try finally block)
- 9 while (while loop with condition)

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("nombres.txt");
15        try {
16            lectorArchivo = new FileReader(archivo);
17            lectorBuffer = new BufferedReader(lectorArchivo);
18
19            nombreEmpresa = lectorBuffer.readLine();
20            while(nombreEmpresa != null) {
21                System.out.println("Nombre de la empresa: " + nombreEmpresa);
22                nombreEmpresa = lectorBuffer.readLine();
23            }
24        } catch (IOException e) {
25            System.out.println("NO se encontró el archivo!");
26            e.printStackTrace();
27        }
28    }
29 }
```

NO se encontró el archivo!  
java.io.FileNotFoundException: nombres.txt (El sistema no puede encontrar el archivo especificado)  
at java.base/java.io.FileInputStream.open0(Native Method)  
at java.base/java.io.FileInputStream.open(FileInputStream.java:219)  
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)  
at java.base/java.io.FileReader.<init>(FileReader.java:75)  
at excepciones.ExcepcionesVerificadas.main(ExcepcionesVerificadas.java:16)

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

The screenshot displays an IDE environment with three main components:

- Left Panel (Code Editor):** Shows the file `ExcepcionesVerificadas.java` with the following code:

```
1 package excepciones;
2
3 import java.io.*;
4
5 public class ExcepcionesVerificadas {
6
7     public static void main(String[] args) {
8         File archivo;
9         FileReader lectorArchivo;
10        BufferedReader lectorBuffer;
11
12        String nombreEmpresa = null;
13
14        archivo = new File("src/excepciones/nombres.txt");
15        try {
16            lectorArchivo = new FileReader(archivo);
17            lectorBuffer = new BufferedReader(lectorArchivo);
18
19            nombreEmpresa = lectorBuffer.readLine();
20            while(nombreEmpresa != null) {
21                System.out.println("Nombre de la empresa: " + nombreEmpresa);
22                nombreEmpresa = lectorBuffer.readLine();
23            }
24        } catch (IOException e) {
25            System.out.println("NO se encontró el archivo!");
26            e.printStackTrace();
27        }
28    }
29 }
```

The `IOException` in the catch block is highlighted with a red rectangle.
- Top Center Panel (Run Output):** Shows the output of the program:

```
1 Compañia Cervecera de Nicaragua
2 Licorera de Nicaragua
3 Tigo
4 Claro
5 Siman
```
- Right Panel (Project Explorer):** Shows the project structure under `ManejoDeExcepciones`:
  - JRE System Library [JavaSE-11]
  - src
    - excepciones
      - ExcepcionesNoVerificadas.java
      - ExcepcionesVerificadas.java
      - nombres.txt

Arrows indicate the flow of data: one arrow points from the `IOException` catch block to the output window, and another points from the `nombres.txt` file in the project explorer to the output window.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Relanzando y lanzando una excepción

- Cuando una excepción ocurre en un bloque **try**, el control pasa inmediatamente al primer bloque **catch** coincidente. Por lo general, un bloque **catch** realiza una de las siguientes acciones:
  - ✓ Maneja por completo la excepción.
  - ✓ Procesa parcialmente la excepción. En este caso, el bloque **catch** relanza la misma excepción o lanza otra excepción para que el entorno solicitante (por ejemplo **main()**) la maneje.
  - ✓ Relanza la misma excepción para que el entorno solicitante (por ejemplo **main()**) la maneje.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Relanzando y lanzando una excepción

- El mecanismo de relanzamiento o lanzamiento de una excepción es muy útil en los casos en que un bloque **catch** atrapa la excepción, pero el bloque **catch** no puede manejar la excepción, o si dicho bloque **catch** decide que la excepción se debe manejar por el entorno solicitante.
- Relanzar o lanzar una excepción se efectúa por la instrucción **throw**. Una instrucción **throw** puede lanzar una excepción verificada o una no verificada.

```
throw referenciaExcepcion;
```

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

RelanzandoExcepcion.java

```
1 package excepciones;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class RelanzandoExcepcion {
7     static Scanner leer = new Scanner(System.in);
8
9     public int obtenerNumero() throws InputMismatchException {
10         int num;
11         try {
12             System.out.print("Ingrese un número entero: ");
13             num = leer.nextInt();
14             System.out.println();
15             return num;
16         } catch (InputMismatchException e) {
17             throw e;
18         }
19     }
20
21     public static void main(String[] args) {
22         int numero;
23         RelanzandoExcepcion obj = new RelanzandoExcepcion();
24         try {
25             numero = obj.obtenerNumero();
26             System.out.println("Número = " + numero);
27         } catch (InputMismatchException e) {
28             System.out.println("Excepción: " + e.toString());
29         }
30     }
31 }
```

Ingrese un número entero: 10

Número = 10

Ingrese un número entero: 5h

Excepción: [java.util.InputMismatchException](#)

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# El Bloque `try...catch...finally`

- Las instrucciones que podrían generar una excepción se colocan en un bloque `try`. Este bloque también podría contener instrucciones que no se deben ejecutar si ocurre una excepción.
- El bloque `try` es seguido por cero o más bloques `catch`. Un bloque `catch` especifica el tipo de excepción que puede atrapar y contiene un manejador de excepciones. El último bloque `catch` puede o no ser seguido por un bloque `finally`.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

El Bloque  
try...catch...finally...

- Cualquier código contenido en un bloque **finally** siempre se ejecuta, sin importar si ocurre una excepción, excepto cuando el programa sale anticipadamente de un bloque **try** invocando al método **System.exit**.
- Si un bloque **try** no tiene bloque **catch**, entonces debe tener el bloque **finally**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# El Bloque try...catch...finally...

- Si no se lanza una excepción en un bloque **try**, todos los bloques **catch** asociados con el bloque **try** se ignoran y la ejecución del programa continúa después del último bloque **catch**.
- Si una excepción se lanza en un bloque **try**, las instrucciones restantes en dicho bloque se ignoran. El programa busca los bloques **catch** en el orden en que aparecen después del bloque **try** y busca un manejador de excepciones apropiado.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

El Bloque  
try...catch...finally...

- Si el tipo de la excepción lanzada coincide con el tipo de parámetro en uno de los bloques **catch**, el código de ese bloque **catch** se ejecuta y los bloques **catch** restantes después de este bloque **catch** se ignoran.
- Si hay un bloque **finally** después del último bloque **catch**, el bloque **finally** se ejecuta sin importar si ocurre una excepción.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Paso de una  
excepción a  
quién lo  
invoque

- Acá se verá la estructura de código necesario cuando no es posible resolver algún problema que surja en algún método.
- Por ejemplo, supongamos que escribimos un método que deba verificar una tarjeta de crédito para autorizar y realizar un pago en esa tarjeta. ¿Qué pasa si la red está caída?
- El método podría intentar resolver pero hasta cierto punto va a ceder y desistir, dependiendo de la lógica del desarrollador.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
public void autorizarTC(Tarjeta C, double monto){  
  
    try{  
  
    } catch(---- -){  
        intentos++;  
        if(intentos>MAX_INTENTOS){  
            throw new ExcepcionParaAutorizacionTC();  
        }  
    }  
}
```

- Supongamos que tenemos el método para verificar tarjeta, y la estructura **try...catch** para intentar resolver en el caso de problemas en la red.
- Entonces se debe crear la lógica para controlar esos intentos. Se puede hacer uso de ciclos **while** para lograrlo.
- **throw** toma como argumento una expresión de tipo **throwable**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
public void autorizarTC(Tarjeta C, double monto){  
  
    try{  
  
    } catch(---- -){  
        intentos++;  
        if(intentos>MAX_INTENTOS){  
            throw new ExcepcionParaAutorizacionTC();  
        }  
    }  
}
```

- Cualquier **throwable** puede ser usado con **throw**. En este ejemplo, se asume que el desarrollador creó la clase de excepción **ExcepcionParaAutorizacionTC()**.
- No tiene que ser así. Podría usarse una clase de excepción preexistente.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
public void autorizarTC(Tarjeta C, double monto){  
  
    try{  
  
    } catch(---- -){  
        intentos++;  
        if(intentos>MAX_INTENTOS){  
            throw new ExcepcionParaAutorizacionTC();  
        }  
    }  
}
```

- Cuando se ejecuta el **throw**, la excepción saldrá del método y reaparecerá donde fue invocado (por ejemplo, **main()**). Ahí, quien lo halla invocado verá lo que pasó.
- En este ejemplo, se ejecuta en el bloque **catch**, no hay algo que restrinja donde podemos hacerlo.
- Si se ejecuta dentro del bloque **catch**, la ejecución saltaría al **catch**, no fuera del método.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
public void autorizarTC(Tarjeta C, double monto)
    throws ExcepcionParaAutorizacionTC {

    try{

    } catch(---- -){
        intentos++;
        if(intentos>MAX_INTENTOS){
            throw new ExcepcionParaAutorizacionTC();
        }
    }
}
```

- Para decirle a quién invoque el método lo que pasó, se puede hacer dentro de la declaración del método usando la expresión **throws**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
public void autorizarTC(Tarjeta C, double monto)
    throws ExcepcionParaAutorizacionTC, otraExcepcionParaAutorizacionTC {

try{

} catch(-----){
    intentos++;
    if(intentos>MAX_INTENTOS){
        throw new ExcepcionParaAutorizacionTC();
    }
}
}
```

- Para decirle a quién invoque el método lo que pasó, se puede hacer dentro de la declaración del método usando la expresión **throws** y el nombre del tipo de clase.
- Podemos separar diferentes tipos de excepciones que podrían ser lanzadas usando una lista separada por coma.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- También es permitido pero no una buena práctica incluir declaraciones para excepciones no verificadas. Ya que este tipo de excepciones quiebran el programa. Si es una excepción no verificada, no es necesario admitir que podría ocurrir.
- Pero si es una excepción verificada y es corregida por el desarrollador, la excepción va a escapar del método, se debe declarar la cláusula **throws** en el método.
- De esta manera, cuando se escribe el código que llama este método, el compilador sabe que puede producirse una excepción, e insistirá que el código que lo llame deba ir dentro de una estructura **try...catch**, o ese método debe contener una sentencia **throws**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

- Asumamos que se tiene una `Excepcion()` y que es una excepción verificada. Si se tiene una sentencia **`throw new Excepcion();`** dentro de un método pero no en un bloque **`try`**, tendríamos que declarar **`throws Excepcion`** en el método, y si la sentencia **`throw`** es ejecutada, el método saldrá a quién lo haya invocado directamente.
- Si la sentencia **`throw new Excepcion();`** está dentro de un bloque **`try`**, pero no hay un bloque **`catch`** asociado con el bloque **`try`**, el **`catch`** es la excepción `Excepcion()` o cualquiera de sus clases padres y el efecto sería el mismo.
- Si la sentencia **`throw new Excepcion();`** está dentro de un bloque **`try`**, y hay un bloque **`catch`** asociado con el bloque **`try`**, la ejecución de la excepción saltará directamente al bloque **`catch`**. En ese punto, Java asume que la excepción ha sido manejada. Desde ese punto en adelante, la ejecución continua normalmente.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Sentencias

# finally para

# limpiar recursos

- **finally** es un mecanismo proveído por el sistema de manejo de excepciones de java, el cual permite limpiar confiablemente los recursos que estén siendo usados por el código.
- Cuando se tiene una estructura de manejo de excepciones como la estudiada anteriormente donde se tiene una ruta en la que todo funciona bien, y se tiene otra ruta donde se intenta corregir problemas o errores que puedan surgir, es normal terminar asignando recursos que necesitan ser liberados.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Sentencias  
finally para  
limpiar recursos

- Java tiene un subsistema llamado Garbage Collector (recolector de basura) que se asegura de que si asignamos memoria y ya no estamos usando esos objetos, libera esos recursos para que el sistema operativo haga uso de ellos. Pero el recolector de basura solo atiende aquellos asuntos que tienen que ver con memoria.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Sentencias finally para limpiar recursos

- Pueden existir otros tipos de recursos que podrían estar disponibles pero con limitaciones a través del sistema operativo.
- Este tipo de recursos puede ser el manejo de archivos. Se usa bloques `try...catch` para lidiar con problemas relacionados con archivos, por ejemplo, no contener los datos correctos, o la estructura de los datos no es la correcta, o el nombre del archivo fue mal indicado, y no se pudo abrir el archivo del todo.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Sentencias  
finally para  
limpiar recursos

- Igual pueden surgir problemas relacionados con la red.
- Se obtendrían excepciones de entrada y salida (I/O) y hay que limpiar esos recursos. En este caso hablamos de sockets, utilizados por el sistema operativo para acceder a recursos de la red. Esto tiene límites de uso también.

# IX. EXCEPCIONES

---

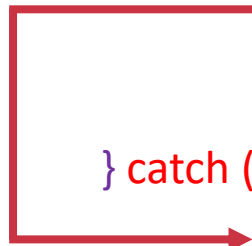
## INTRODUCCIÓN A LAS EXCEPCIONES

- Igual pueden surgir problemas relacionados con la red.
- Se obtendrían excepciones de entrada y salida (I/O) y hay que limpiar esos recursos. En este caso hablamos de sockets, utilizados por el sistema operativo para acceder a recursos de la red. Esto tiene límites de uso también.
- Java tiene un mecanismo para liberar esos tipos de recursos, usando el bloque **finally**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

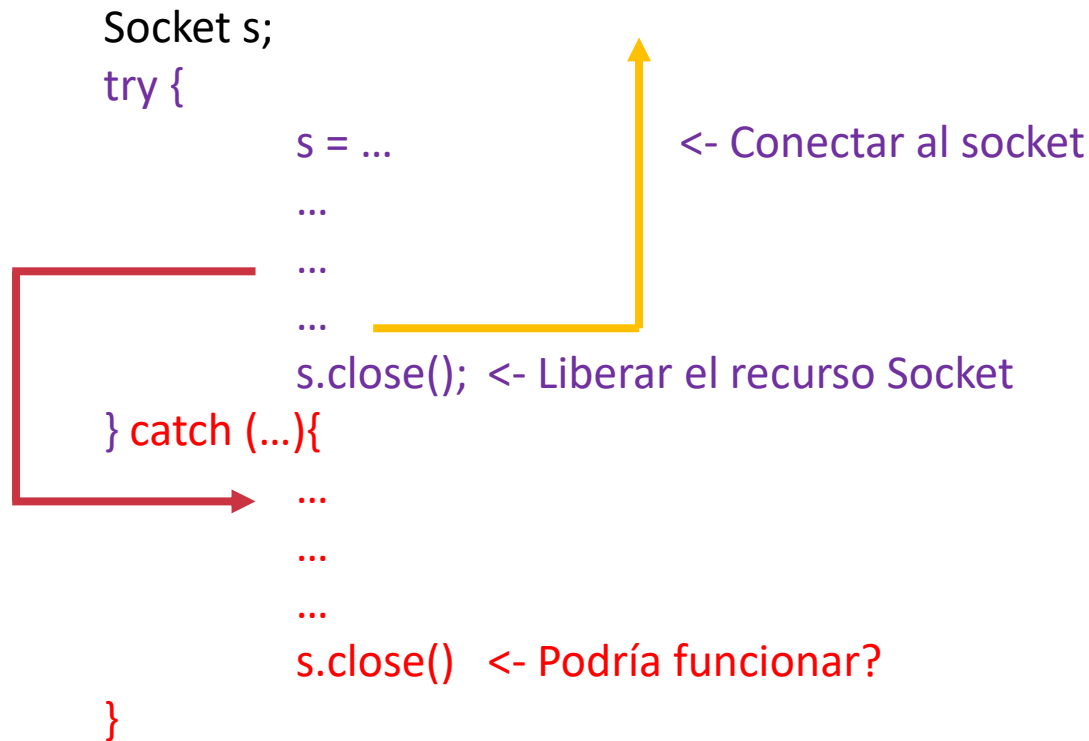
```
Socket s;  
try {  
    s = ...           <- Conectar al socket  
    ...  
    ...  
    ...  
    s.close(); <- Liberar el recurso Socket  
} catch (...){  
    ...  
    ...  
    ...  
}
```



Esto podría crea una fuga (leak) de Socket, no de memoria. Si el programa ha estado ejecutándose por mucho tiempo podría consumirse todos los recursos de Socket del sistema operativo, entonces el sistema operativo va a matar nuestro programa.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES



Se podría poner una sentencia de cierre del Socket dentro del bloque **catch**. Pero si se tiene múltiples bloques **catch**, tendríamos que duplicar la sentencia de cierre del Socket. Pero hacer eso no garantiza que se resuelva el problema, porque está la posibilidad de poder salir del programa con una **excepción no verificada**, o al menos una para la cual no especificamos un bloque **catch**.

# IX. EXCEPCIONES

---

## INTRODUCCIÓN A LAS EXCEPCIONES

- Lo que se necesita es un mecanismo que garantice que tan pronto como se abra el Socket se podrá cerrar confiablemente, sin importar cómo salga del código.
- Java provee este mecanismo a través del bloque **finally**.

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

```
Socket s = null;
```

```
try {
```

```
    s = ...
```

<- Conectar al socket

```
    ...
```

```
    ...
```

```
    ...
```

```
    } catch (...){
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    } finally {
```

```
        if(s != null)
```

s.close(); <- Liberar el recurso Socket

```
    }
```

Si se entra al bloque **try**, luego se ejecuta el bloque **finally**. En este escenario, todo funciona correctamente.

Si se entra al bloque **try**, y se genera un excepción, se ejecuta el bloque **catch**. Después del **catch** se ejecutará el bloque **finally**. En este escenario, todo funciona correctamente.

Si se entra al bloque **try**, y se genera un excepción para la cual no hay una excepción disponible, se ejecutará el bloque **finally**. En este escenario, todo funciona correctamente.

En cualquiera de esos casos, se ejecutará el bloque **finally**. Claro, si el abastecimiento energético es interrumpido, posiblemente no se ejecute el bloque **finally**. Otro caso especial podría ser si se genera una excepción dentro del bloque **finally**. Se puede agregar una estructura **try ... catch ... finally...** dentro del bloque **finally**.



# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

Existen otras  
formas de  
manejar  
excepciones

```
Socket s = null;  
finally {  
    try {  
        if(s != null)  
            s.close();  
    } catch (IOException ioe){  
        System.out.println(Cierre fallido);  
    }  
}
```

---

```
try (Socket s = x.Connect();  
    InputStream is = y.open());{  
    ...  
    ...  
    ...  
} catch (...){  
    ...  
}
```

↑  
Es opcional

# IX. EXCEPCIONES

## INTRODUCCIÓN A LAS EXCEPCIONES

# Variantes posibles del uso de manejo de excepciones

try + catch  
try + finally  
try + catch + finally

~~try~~  
~~catch~~  
~~finally~~

try

try con recursos [try(var v = ...)...]  
Implica try + finally