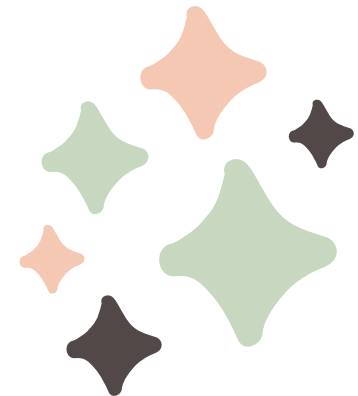


# Excepciones en Java

DANNA VALERIA MORALES AGUILAR



# Introducción ✨

Las excepciones son situaciones anormales que pueden ocurrir durante la ejecución de las aplicaciones, en algunos casos no se pueden gestionar como los de la propia JVM.

Java proporciona un mecanismo de gestión de excepciones cuando estemos obligados a gestionarlas



```
int[] array = new int[10];  
System.out.println(array[10]);
```

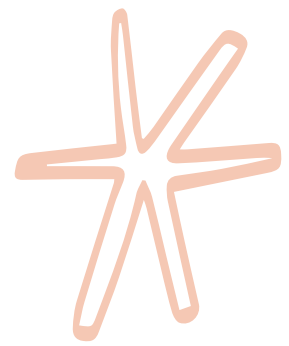
# ¿Qué es una excepción?

Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de ejecución. Las excepciones son utilizadas para manejar errores y situaciones excepcionales de manera estructurada.

# ¿QUÉ ES UN ERROR?

```
<terminated> OWLAPITutorial [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java (19/8/2015 11:43:08)
Exception in thread "main" java.lang.NoClassDefFoundError: com/google/inject/Provider
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:760)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:455)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:367)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:360)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:760)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:455)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:367)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
    at java.security.AccessController.doPrivileged(Native Method)
```

Un error es una subclase que indica problemas graves en la aplicación. Este problema no puede ser resultado de ninguna manera por lo que normalmente el programa se detiene.



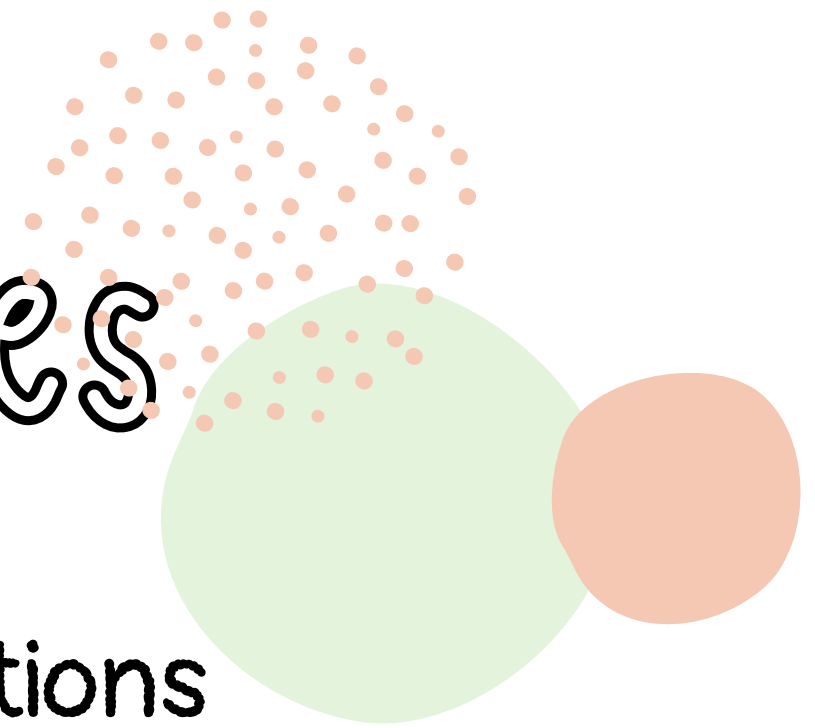
# Tipos de Excepciones

## Runtime Exceptions

Las excepciones de tiempo de ejecución son un subtipo de excepciones no verificadas que se derivan de la clase "RuntimeException". No están obligadas a ser capturadas mediante la cláusula "throws". Indican errores de programación.

## IOExceptions

Las excepciones de Entrada/Salida son un subtipo de excepciones verificadas. Están obligadas a ser capturadas mediante la cláusula "throws". Ocurren cuando hay un problema con las operaciones de entrada/salida, como lectura o escritura de archivos, conexiones de red, etc.





# Ejemplos de Runtime Exceptions

## 01 NullPointerException

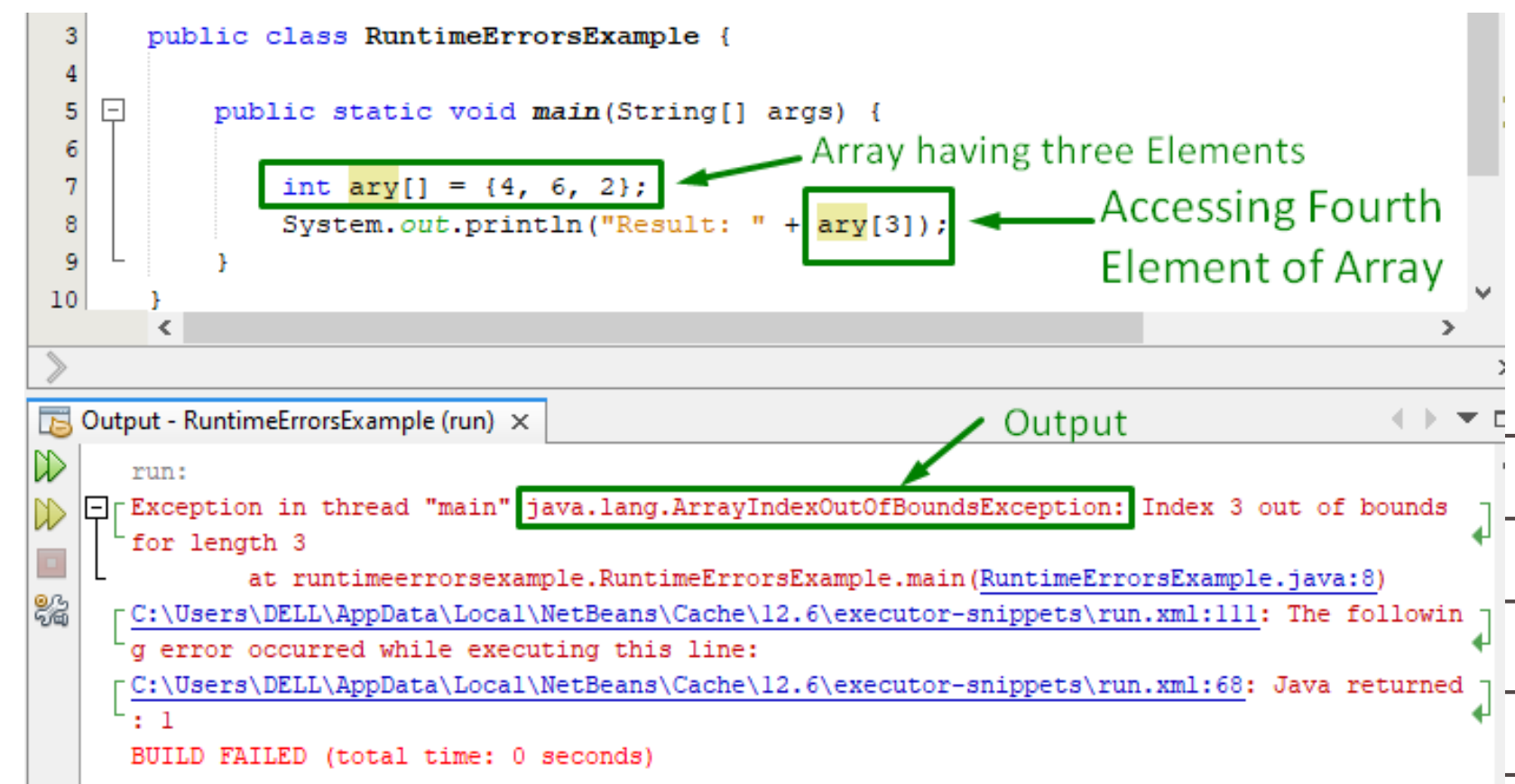
Ocorre cuando se intenta acceder a un objeto o llamar a un método en una variable de referencia nula.

## 02 ArrayIndexOutOfBoundsException

Se lanza cuando se intenta acceder a un índice inválido en un arreglo.

## 03 ArithmeticException

Se produce cuando ocurre una operación aritmética inválida, como la división por cero



```
3 public class RuntimeErrorsExample {
4
5     public static void main(String[] args) {
6
7         int ary[] = {4, 6, 2};
8         System.out.println("Result: " + ary[3]);
9     }
10 }
```

Array having three Elements

Accessing Fourth Element of Array

Output - RuntimeErrorsExample (run) x

run:

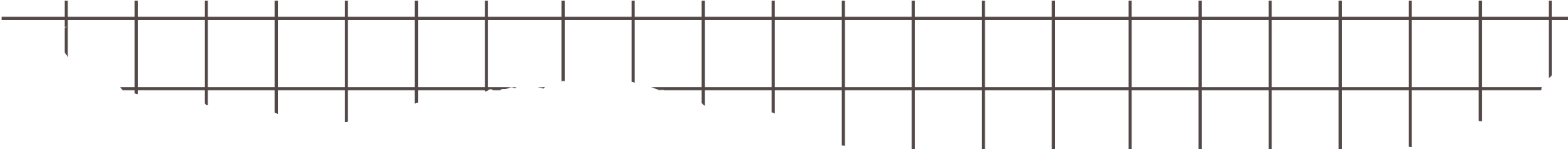
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3

at runtimeerrorsexample.RuntimeErrorsExample.main(RuntimeErrorsExample.java:8)

[C:\Users\DELL\AppData\Local\NetBeans\Cache\12.6\executor-snippets\run.xml:111: The following error occurred while executing this line:

[C:\Users\DELL\AppData\Local\NetBeans\Cache\12.6\executor-snippets\run.xml:68: Java returned : 1

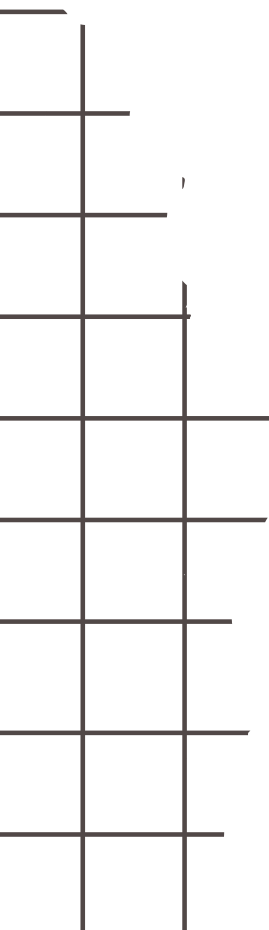
BUILD FAILED (total time: 0 seconds)



**04** **ClassCastException**  
Ocurre cuando se intenta realizar una conversión de tipo inválida.

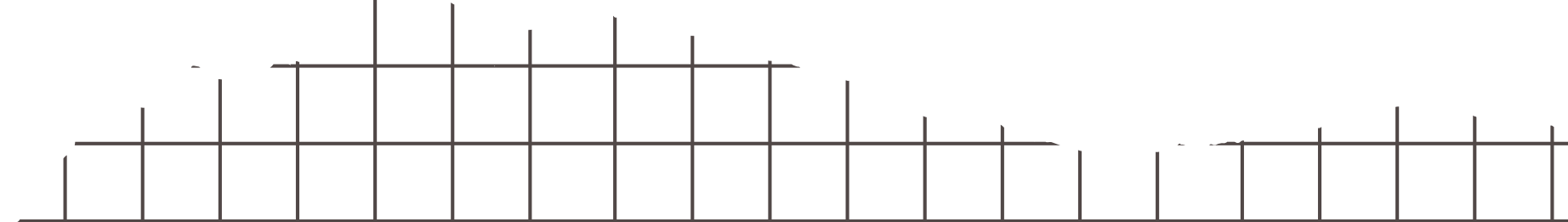


**IllegalArgumentException** **05**  
Se lanza cuando un método recibe un argumento inválido.



**06** **NumberFormatException**  
Se produce cuando se intenta convertir una cadena a un número, pero la cadena no tiene el formato adecuado.

**IndexOutOfBoundsException** **07**  
Es una superclase de **ArrayIndexOutOfBoundsException**, y se lanza cuando se intenta acceder a un índice inválido en estructuras de datos como listas o colecciones.



# Ejemplos de IOExceptions

## 01 FileNotFoundException

Se lanza cuando se intenta acceder a un archivo que no existe.

## IOException

Es la clase base para excepciones relacionadas con problemas de entrada/salida en general.

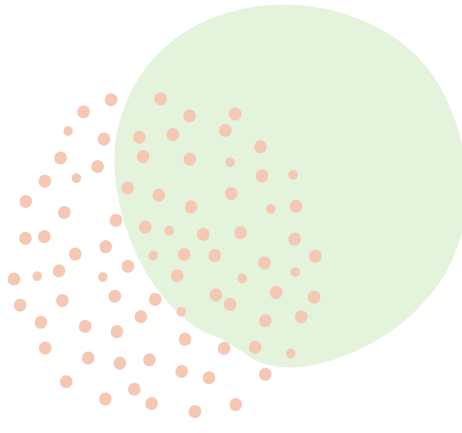
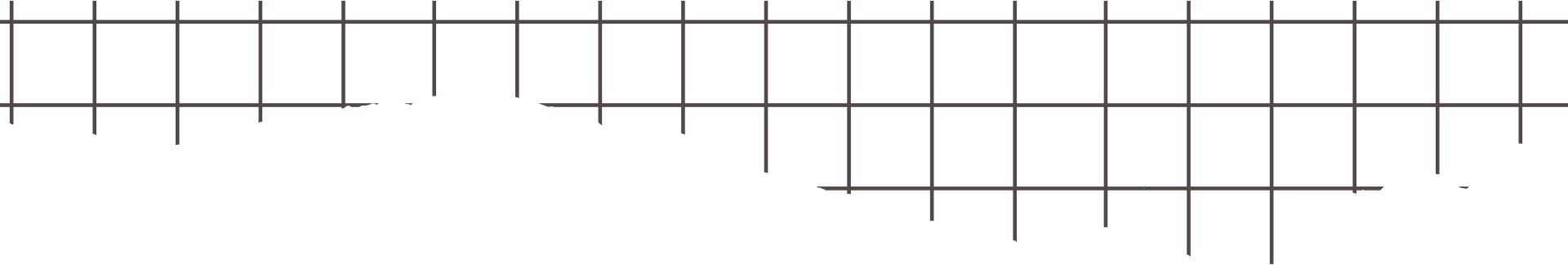
## 03 SocketException

Se lanza cuando ocurre un problema en una operación de socket, como una conexión de red fallida.

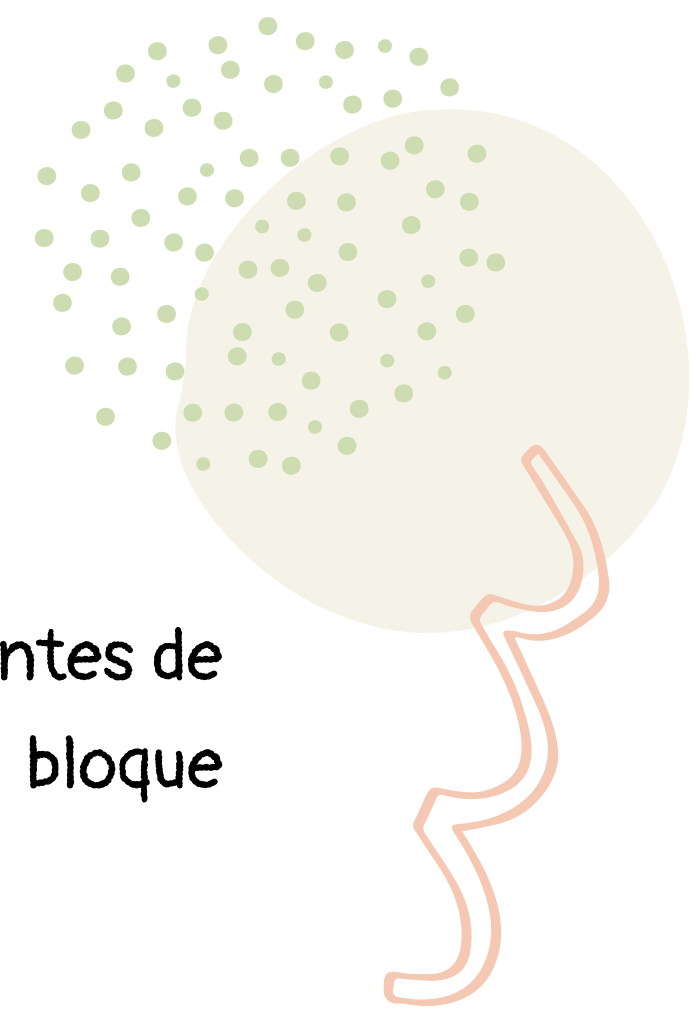
## EOFException

Se produce cuando se alcanza el final de un flujo de entrada antes de que se complete una lectura completa.





# Java MultiCatch Block




Es un bloque de código en el que se capturan varias excepciones problemáticas. Antes de Java 7 para capturar múltiples tipos de excepciones era necesario repetir el bloque "catch" para cada tipo de excepción, lo que generaba código redundante.

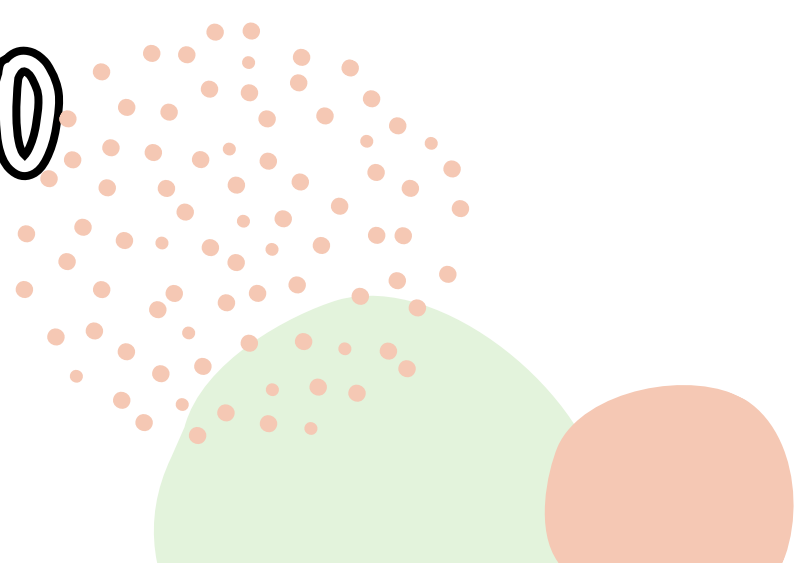
Ahora se puede capturar más de un tipo de excepción separándolos por barras verticales (|) en el bloque catch.

```
try {  
    // Código que puede lanzar diferentes excepciones  
} catch (IOException | SQLException e) {  
    // Manejo de excepciones IOException y SQLException  
}
```

# Explicación de código



Cree una excepción personalizada llamada "PostreException" que hereda de la clase "Exception". Esta excepción se lanzará cuando se disfrute de un postre generico, los cuales serian la clase "Postre". Las clases "Pastel" y "Helado" sobre escriben el método "disfrutar()", cuando se ejecuta este método cuando se disfruta un pastel o un helado, pero cuando se intenta disfrutar de un postre genérico se lanza la excepción "PostreException":



```
1 package MC;
2
3 public class PostreException extends Exception{
4
5     public PostreException(String mensaje) {
6         super(mensaje);
7     }
8 }
9
```

```
1 package MC;
2
3 public class Postre {
4     void disfrutar() throws PostreException {
5         throw new PostreException("Error al disfrutar el postre");
6     }
7 }
8
```

```

1 package MC;
2
3 public class Pastel extends Postre{
4     void disfrutar() throws PostreException {
5         System.out.println("El pastel es increíble");
6     }
7 }

```

```

1 package MC;
2
3 public class Helado extends Postre{
4     void disfrutar() throws PostreException {
5         System.out.println("El helado es refrescante");
6     }
7 }

```

```

public class Public {

    public static void main(String[] args) {
        try {
            Postre pastel = new Pastel();
            Postre helado = new Helado();

            //Postre generico
            Postre pay = new Postre();

            pastel.disfrutar();
            helado.disfrutar();

            pay.disfrutar();
        } catch (PostreException e) {
            e.printStackTrace();
            System.out.println("Se produjo una excepción");
        }
        System.out.println("Termina programa");
    }
}

```

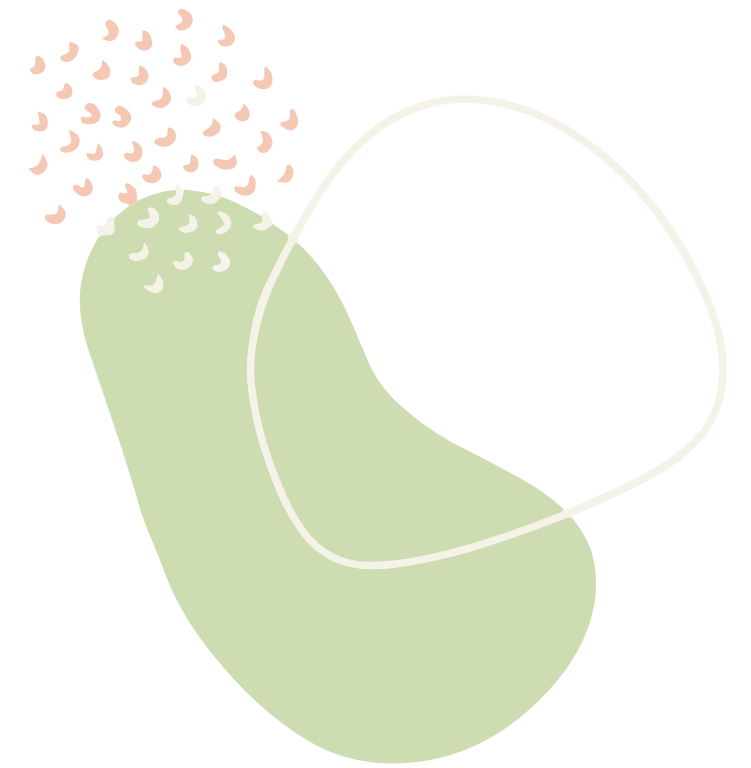
## Resultado con Excepcion

```
El pastel es increíble  
El helado es refrescante  
MC.PostreException: Error al disfrutar el postre  
    at MC.Postre.disfrutar(Postre.java:5)  
Se produjo una excepción  
    at MC.Public.main(Public.java:13)  
Termina programa
```

## Resultado sin Excepcion

```
El pastel es increíble  
El helado es refrescante  
Termina programa
```

# try-with-resources



Su objetivo es cerrar los recursos de forma automática en la sentencia try-catch-finally y hacer mas simple el código. Garantiza que los recursos abiertos se cierren automáticamente al salir del bloque "try", ya sea que el bloque se complete normalmente o que se lance una excepcion.

```
try (RecursoDeclarado recurso = new RecursoDeclarado()) {  
    // Código que utiliza el recurso  
} catch (Excepcion e) {  
    // Manejo de excepciones (opcional)  
}
```

# Explicación de código\*

Lo que esta haciendo este código es realizar una solicitud a una página web sobre conceptos básicos de Java. El código se conecta al sitio web, obtiene el contenido que se tiene en la página y lo muestra en la salida de la consola.

El uso de la sentencia try-with-resources garantiza que `BufferedReader` se cierre correctamente y automáticamente al salir del bloque, esto asegura una gestión adecuada de los recursos de conexión sin la necesidad de escribir un bloque "finally" para cerrarlo manualmente.





```
package twr;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class Principal {

    public static void main(String[] args) {
        try {
            URL url = new URL("https://www.manualweb.net/java/conceptos-basicos-java/");
            HttpURLConnection con = (HttpURLConnection) url.openConnection();

            // Leer la respuesta del servidor
            try (BufferedReader br = new BufferedReader(new InputStreamReader(con.getInputStream()))) {
                String inputLine;
                while ((inputLine = br.readLine()) != null) {
                    System.out.println(inputLine);
                }
            }
        } catch (IOException e) {
            System.err.println("Error al realizar la conexión: " + e.getMessage());
        }
    }
}
```