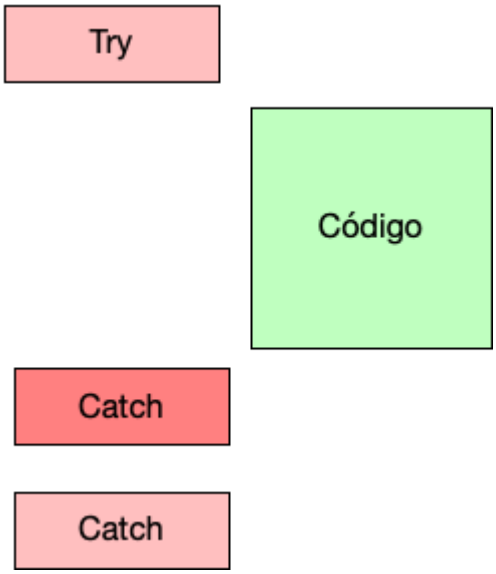
El concepto de Java MultiCatch Block es uno de los conceptos más clásicos de Manejo de Excepciones. En un bloque de código podemos tener la necesidad de Capturar varias Excepciones problemáticas.

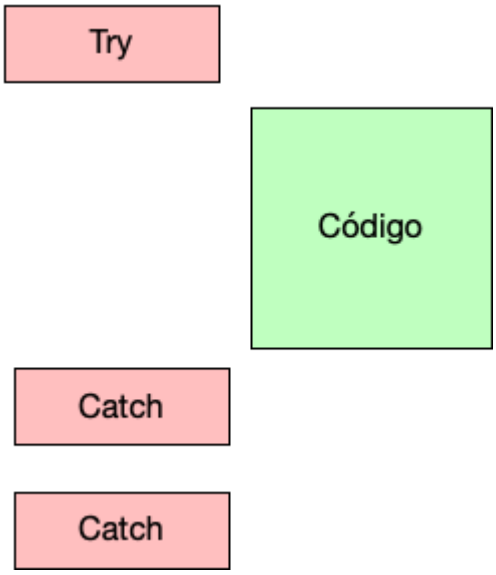


Por ejemplo podemos tener un código que pueda lanzar una Excepción de base de datos así como una Excepción de ficheros.

```
package com.arquitecturajava;

import java.io.FileWriter;
import java.io.IOException;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Principal {

        public static void main(String[] args) {

                FileWriter fw = null;
                try {
```

```
                        fw = new FileWriter("nuevo.txt");
                        fw.write("hola soy un fichero de texto");
DriverManager.getConnection("jdbc:mysql://localhost:3306", "root",
"root");
                        System.out.println("fichero creado");
                        System.out.println("acceso a base de datos");
                } catch (IOException e1) {

                        System.out.println("error de fichero"+e1);

                } catch (SQLException e1) {
                        System.out.println("error de SQL"+e1);

                }

        }

}
```

Hasta aquí todo correcto . Pero en muchos casos sucede que un conjunto de clausulas Catch comparten la misma gestión de excepciones . Es decir en nuestro primer ejemplo cada excepción es tratada de una forma especial pero hay muchos casos que las excepciones se tratan del mismo modo .



Por ejemplo:

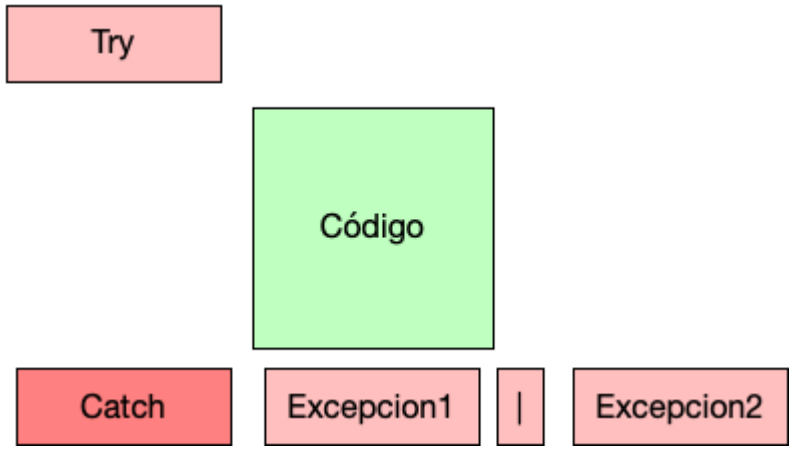
```
package com.arquitecturajava;

import java.io.FileWriter;
import java.io.IOException;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Principal {

        public static void main(String[] args) {

                FileWriter fw = null;
                try {
                        fw = new FileWriter("nuevo.txt");
                        fw.write("hola soy un fichero de texto");
DriverManager.getConnection("jdbc:mysql://localhost:3306", "root",
"root");
                        System.out.println("fichero creado");
```

```
                System.out.println("acceso a base de datos");
            } catch (IOException e1) {

                System.out.println("error "+e1);

            } catch (SQLException e1) {
                System.out.println("error "+e1);

            }

        }

}
```

En este caso tenemos dos Excepciones que se tratan de la misma forma y cada una de ellas tiene definida su propia clausula catch . Es decir tenemos un java multicatch block pero gestionamos las Excepciones de la misma manera.

Java MultiCatch Block y Java 7

A partir de Java 7 se permite compactar este tipo de manejo de Excepciones en una única linea agrupando múltiples Excepciones de forma conjunta ya que asumimos que su tratamiento será idéntico .




```
package com.arquitecturajava;

import java.io.FileWriter;
import java.io.IOException;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Principal {

        public static void main(String[] args) {

                FileWriter fw = null;
                try {
                        fw = new FileWriter("nuevo.txt");
                        fw.write("hola soy un fichero de texto");
DriverManager.getConnection("jdbc:mysql://localhost:3306", "root",
"root");
                        System.out.println("fichero creado");
                        System.out.println("acceso a base de datos");
                } catch (IOException | SQLException e1) {

                        System.out.println("error "+e1);
```

```
                }
        }

}
```

En este caso:

Acabamos de usar un Java MultiCatch Block compacto que nos permite ver el código de una forma más limpia y práctica sin perder funcionalidad a nivel de gestión de errores.

Otros artículos relacionados

- Java Try Catch y su manejo
- Java Finally y el cierre de recursos
- JDBC, Java try with resources
- Java Thread concurrencia y Runnables
- Java Exception