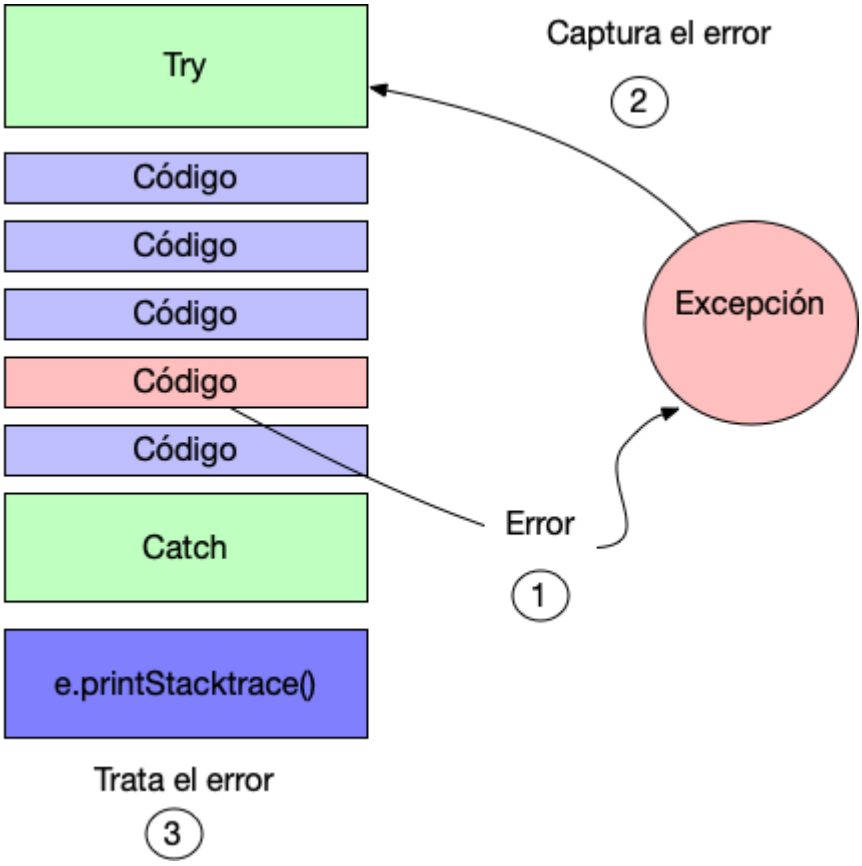
El uso de Java Try Catch como claúsula para gestionar el manejo de excepciones es una de las cosas más habituales en el lenguaje . Cualquier bloque de código puede encontrarse con una situación excepcional y lanzar un mensaje de error. Estos mensajes de error están definidos por Excepciones clases que almacenan toda la información sobre un error y nos permiten un acceso sencillo a ello. Vamos a ver unos sencillos ejemplos sobre su manejo apoyándonos en código de JDBC:

```
package com.arquitecturajava.excepciones;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class Principal {

  public static void main(String[] args) {

    Connection conexion;
    String url = "jdbc:mysql://localhost:3306/biblioteca";
    String usuario = "root";
    String clave = "";
    String consulta = "insert into Libros
(isbn,titulo,autor,precio,categoria) values
('5','net','juan',20,'web')";

      try {
        Class.forName("com.mysql.jdbc.Driver");
        conexion = DriverManager.getConnection(url, usuario, clave);
        Statement sentencia = conexion.createStatement();
        sentencia.execute(consulta);
```

```
        } catch (ClassNotFoundException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        } catch (SQLException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }
    }

}
```

En este bloque de código nos encontramos con que se pueden producir varias situaciones excepcionales

1. La primera es que no encontremos el Driver de conexión a la base de datos que es una clase com.mysql.jdbc.Driver
2. El segundo es que la consulta no se pueda ejecutar contra la base de datos porque tenga algún tipo de problema , este mal construida , no exista la tabla etc.
   Ambas situaciones son excepcionales y es probable que nuestro programa no se pueda recuperar fácilmente del error. Para solventar este problema se una un Java Try Catch (Bloque Try/Catch) que permite que el programa intente ejecutar el código y si encuentra algún tipo de error lo capture (cláusula catch) y proceda a gestionarlo.



De esta manera podemos recuperar la ejecución normal del programa y avisar de cualquier error que se produzca.

Java Try Catch y multiples cláusulas

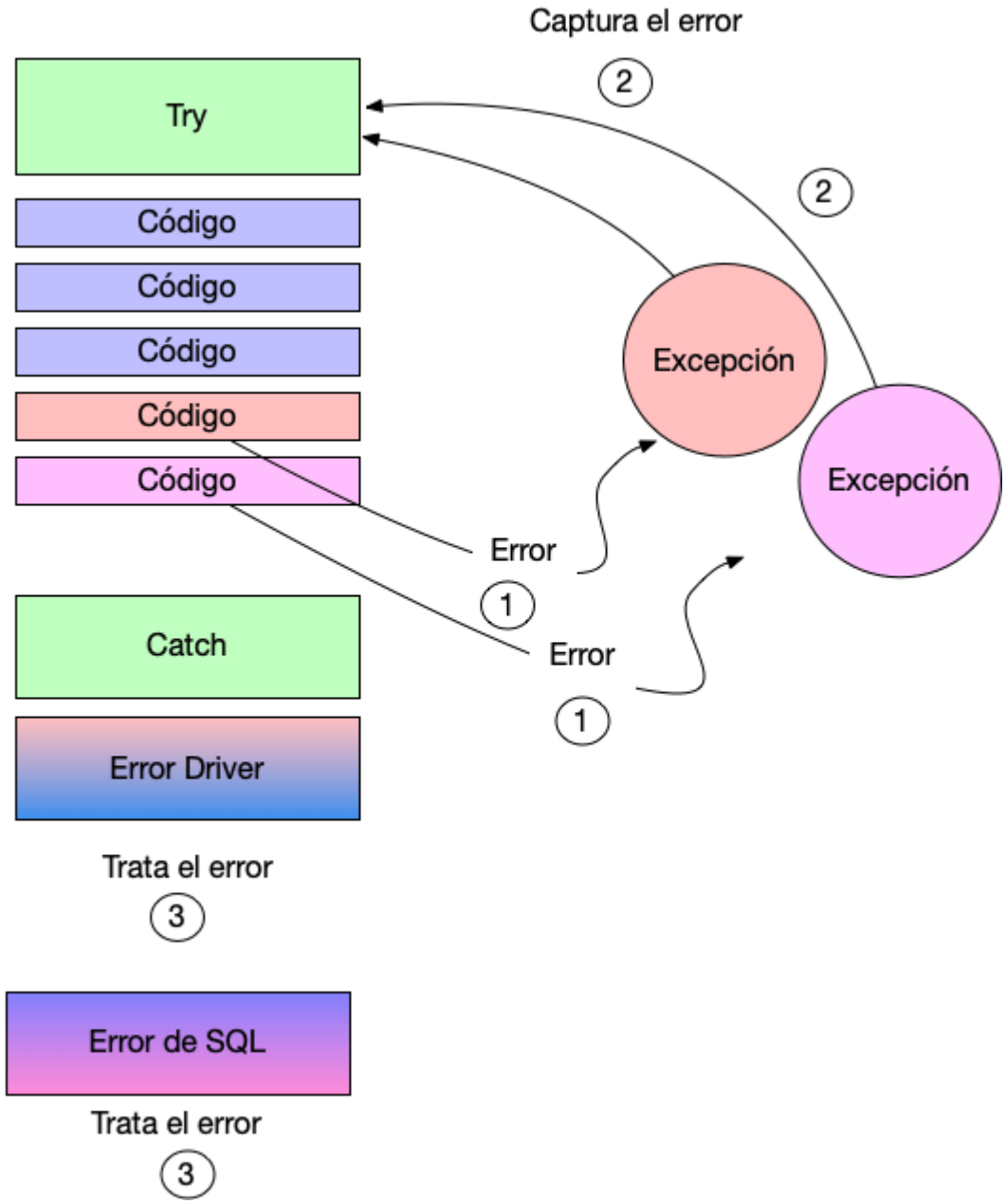
El uso del bloque Try/Catch no esta limitado a una única captura de la excepción sino que podemos tener diferentes bloques try/catch dependiendo del error que se produzca permitiéndonos en cada caso imprimir un mensaje de error diferente.

```
package com.arquitecturajava.excepciones;

import java.sql.Connection;
import java.sql.DriverManager;
```

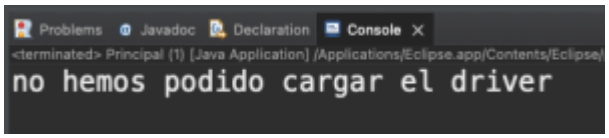
```
import java.sql.SQLException;
import java.sql.Statement;

public class Principal {

  public static void main(String[] args) {

    Connection conexion;
    String url = "jdbc:mysql://localhost:3306/biblioteca";
    String usuario = "root";
    String clave = "";
    String consulta = "insert into Libros
(isbn,titulo,autor,precio,categoria) values
('5','net','juan',20,'web')";

      try {
        Class.forName("com.mysql.jdbc.Driver");
        conexion = DriverManager.getConnection(url, usuario, clave);
        Statement sentencia = conexion.createStatement();
        sentencia.execute(consulta);
      } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        System.out.println("no hemos podido cargar el driver");
      } catch (SQLException e) {
        System.out.println("la consulta SQL es erronea");
      }
  }

}
```

En este caso un bloque Try/Catch avisa de que no hemos podido cargar el Driver de conexión a la base de datos mientras que el segundo bloque avisa de que la consulta SQL

que hemos intentado lanzar es incorrecta. Cada clausula catch gestiona usa casuística diferente.



En este caso si ejecutamos el código nos daremos cuenta de que al no haber instalado el Driver



El programa podrá continuar con su ejecución de forma natural

Otros artículos relacionados

1. JDBC, Java try with resources
2. Java 8 Optional y NullPointerExceptions
3. Java Herencia vs Interfaces