

Curso de Java a distancia

Clase 28: Excepciones en Java - try/catch

Java dispone de un mecanismo de capturar (catch) ciertos tipos de errores que solo pueden ser detectados en tiempo de ejecución del programa.

Los ejemplos más comunes que podemos nombrar de excepciones:

- Tratar de convertir a entero un String que no contiene valores numéricos.
- Tratar de dividir por cero.
- Abrir un archivo de texto inexistente o que se encuentra bloqueado por otra aplicación.
- Conectar con un servidor de bases de datos que no se encuentra activo.
- Acceder a subíndices de vectores y matrices fuera de rango.

La captura de excepciones nos permite crear programas mucho más robustos y tolerante a fallas que ocurren en escasas situaciones, pero en caso que se presenten disponemos de un algoritmo alternativo para reaccionar a dicha situación evitando que el programa finalice su ejecución.

Veremos un ejemplo sin captura de excepciones y luego la sintaxis implementando su captura.

Problema:


Realizar la carga de un número entero por teclado e imprimir su cuadrado.

Programa:

```
import java.util.Scanner;

public class CuadradoNumero {
    public static void main(String[] ar) {
        Scanner teclado = new Scanner(System.in);
        int num;
        System.out.print("Ingrese un valor entero:");
        num = teclado.nextInt();
        int cuadrado = num * num;
        System.out.print("El cuadrado de " + num + " es " + cuadrado);
    }
}
```

Si ejecutamos el programa y el operador ingresa un String en lugar de un entero se genera una excepción y el programa se detiene en forma inmediata:

The screenshot shows an IDE with a file named 'CuadradoNumero.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 public class CuadradoNumero {
4     public static void main(String[] ar) {
5         Scanner teclado = new Scanner(System.in);
6         int num;
7         System.out.print("Ingrese un valor entero:");
8         num = teclado.nextInt();
9         int cuadrado = num * num;
10        System.out.print("El cuadrado de " + num + " es " + cuadrado);
11    }
12 }
```

The console output shows the program terminated and the prompt 'Ingrese un valor entero:' followed by the input 'hola'. An exception is thrown: 'java.util.InputMismatchException'. The stack trace is as follows:

```
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at CuadradoNumero.main(CuadradoNumero.java:8)
```

El programa no ejecuta más instrucciones después de la línea 8 del método main. Se informa el nombre de excepción generada: java.util.InputMismatchException, es decir se nos informa el nombre de clase de excepción y el paquete que la contiene: java.util

Lo primero que podemos decir es que éste tipo de excepciones no son obligatorias capturarlas, pero como vemos si nuestro usuario del programa en forma frecuente se equivoca y carga cadenas en lugar de enteros el programa se detiene y requiere que lo vuelva a ejecutar.

Podemos implementar un programa más robusto si capturamos la excepción InputMismatchException y le informamos al usuario que debe ingresar un entero obligatoriamente.

Veamos ahora la sintaxis del mismo programa pero capturando la excepción:

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class CuadradoNumero {
    public static void main(String[] ar) {
        Scanner teclado = new Scanner(System.in);
        int num;
        try {
            System.out.print("Ingrese un valor entero:");
            num = teclado.nextInt();
            int cuadrado = num * num;
            System.out.print("El cuadrado de " + num + " es " + cuadrado);
        } catch (InputMismatchException ex) {
            System.out.println("Debe ingresar obligatoriamente un número entero.");
        }
    }
}
```

Para atrapar las excepciones debemos encerrar en un bloque try las instrucciones que generan excepciones, en nuestro caso el método 'nextInt' de la clase Scanner:

```
try {
    System.out.print("Ingrese un valor entero:");
    num = teclado.nextInt();
}
```

```

    int cuadrado = num * num;
    System.out.print("El cuadrado de " + num + " es " + cuadrado);
}

```

Todo bloque try requiere que sea seguido por un bloque catch:

```

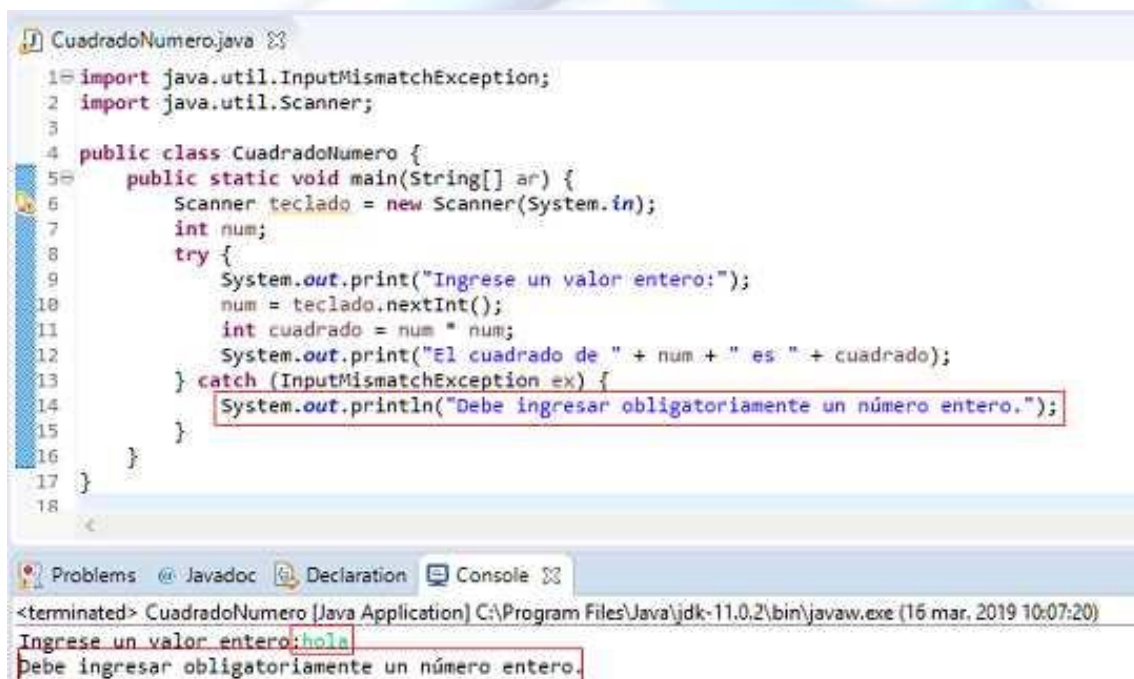
catch (InputMismatchException ex) {
    System.out.println("Debe ingresar obligatoriamente un número entero.");
}

```

Luego de la palabra clave catch se indica entre paréntesis el nombre de un parámetro cualquiera (en nuestro caso lo llamamos 'ex') y el nombre de la excepción a capturar.

El bloque catch normalmente no se ejecuta salvo en los casos excepcionales que dentro del bloque try informa que se disparó dicha excepción.

Ahora si ejecutamos el programa y el operador ingresa un String en lugar de un entero nuestro programa no se detiene sino se le informa el tipo de error que cometió el usuario:



```

CuadradoNumero.java
1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 public class CuadradoNumero {
5     public static void main(String[] ar) {
6         Scanner teclado = new Scanner(System.in);
7         int num;
8         try {
9             System.out.print("Ingrese un valor entero:");
10            num = teclado.nextInt();
11            int cuadrado = num * num;
12            System.out.print("El cuadrado de " + num + " es " + cuadrado);
13        } catch (InputMismatchException ex) {
14            System.out.println("Debe ingresar obligatoriamente un número entero.");
15        }
16    }
17 }
18
Problems  Javadoc  Declaration  Console
<terminated> CuadradoNumero [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 mar. 2019 10:07:20)
Ingrese un valor entero:hola
Debe ingresar obligatoriamente un número entero.

```

Cuando se dispara una excepción las instrucciones que hay luego del método que la disparó no se ejecutan:

```

    num = teclado.nextInt();
    int cuadrado = num * num;
    System.out.print("El cuadrado de " + num + " es " + cuadrado);

```

La dos instrucciones luego de llamar al método 'nextInt' no se ejecutan si dicho método eleva la excepción.

Podemos modificar ahora nuestro programa aun más para que no finalice hasta que cargue un valor entero:

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class CuadradoNumero {
    public static void main(String[] ar) {

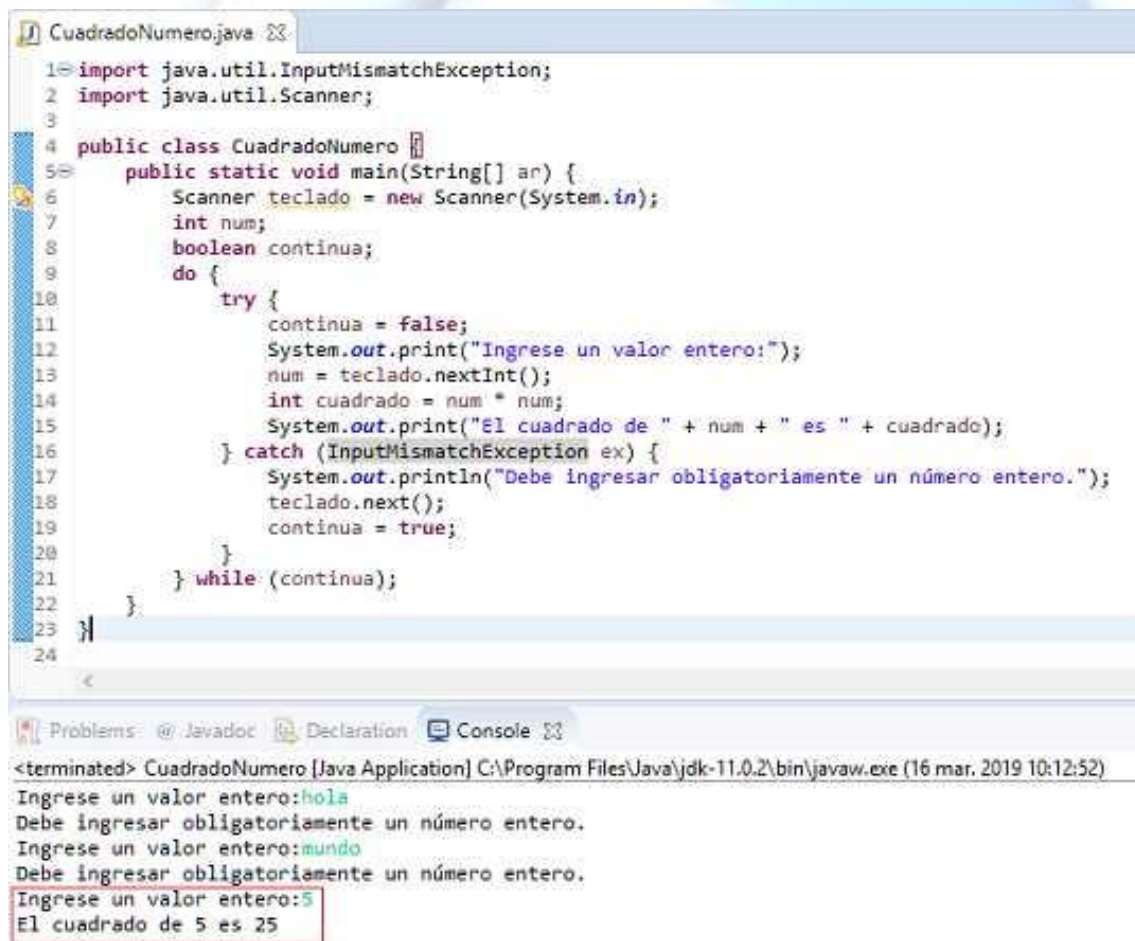
```

```

Scanner teclado = new Scanner(System.in);
int num;
boolean continua;
do {
    try {
        continua = false;
        System.out.print("Ingrese un valor entero:");
        num = teclado.nextInt();
        int cuadrado = num * num;
        System.out.print("El cuadrado de " + num + " es " + cuadrado);
    } catch (InputMismatchException ex) {
        System.out.println("Debe ingresar obligatoriamente un número entero.");
        teclado.next();
        continua = true;
    }
} while (continua);
}

```

Si ejecutamos esta variante de programa, el pedido de la carga del entero no finalizará hasta que lo ingrese:



```

CuadradoNumero.java
1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 public class CuadradoNumero {
5     public static void main(String[] ar) {
6         Scanner teclado = new Scanner(System.in);
7         int num;
8         boolean continua;
9         do {
10             try {
11                 continua = false;
12                 System.out.print("Ingrese un valor entero:");
13                 num = teclado.nextInt();
14                 int cuadrado = num * num;
15                 System.out.print("El cuadrado de " + num + " es " + cuadrado);
16             } catch (InputMismatchException ex) {
17                 System.out.println("Debe ingresar obligatoriamente un número entero.");
18                 teclado.next();
19                 continua = true;
20             }
21         } while (continua);
22     }
23 }
24
Problems @ Javadoc Declaration Console
<terminated> CuadradoNumero [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 mar. 2019 10:12:52)
Ingrese un valor entero:hola
Debe ingresar obligatoriamente un número entero.
Ingrese un valor entero:mundo
Debe ingresar obligatoriamente un número entero.
Ingrese un valor entero:5
El cuadrado de 5 es 25

```

Excepciones - múltiples catch para un try

Podemos definir varios bloques catch para un solo bloque try. Es común que en un bloque try haya más de un método que pueda elevar excepciones o inclusive un mismo método puede generar más de un tipo de excepción.

Luego podemos disponer una sintaxis de try/catch:

```
try {  
    [instrucciones 1]  
} catch([excepción 1]) {  
    [instrucciones 2]  
}  
catch([excepción 2]) {  
    [instrucciones 3]  
}  
catch([excepción n]) {  
    [instrucciones n]  
}
```

Problema:

Realizar la carga de 2 enteros por teclado, mostrar el resultado de dividir el primero por el segundo.

Programa:

```
import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class DivisionEnteros {  
    public static void main(String[] ar) {  
        Scanner teclado = new Scanner(System.in);  
        try {  
            int num1, num2;  
            System.out.print("Ingrese primer valor entero (dividendo):");  
            num1 = teclado.nextInt();  
            System.out.print("Ingrese segundo valor entero (divisor):");  
            num2 = teclado.nextInt();  
            int resu = num1 / num2;  
            System.out.print("La división de " + num1 + " / " + num2 + " es " + resu);  
        } catch (InputMismatchException ex) {  
            System.out.println("Debe ingresar obligatoriamente números enteros");  
        } catch (ArithmeticException ex) {  
            System.out.println("No se puede dividir por cero");  
        }  
    }  
}
```

El bloque try implementa la captura de la excepción 'InputMismatchException' que puede suceder en cualquiera de las dos llamadas al método 'nextInt' de la clase Scanner. También implementa la captura de la excepción 'ArithmeticException' que sucede si en la línea siguiente la variable num2 almacena un cero:

```
int resu = num1 / num2;
```


Problema:

Confeccionar una aplicación visual que permita ingresar en controles de tipo JTextField dos valores enteros. Al presionar un botón mostrar la división del primero respecto al segundo en el título del JFrame o un mensaje si no se ingresan datos o la división no se puede efectuar.

Programa:

```
import javax.swing.*.*;
import java.awt.event.*;

public class Formulario extends JFrame implements ActionListener {
    private JTextField textfield1, textfield2;
    private JButton boton1;

    public Formulario() {
        setLayout(null);
        textfield1 = new JTextField();
        textfield1.setBounds(120, 10, 150, 20);
        add(textfield1);
        textfield2 = new JTextField();
        textfield2.setBounds(120, 40, 150, 20);
        add(textfield2);

        boton1 = new JButton("Dividir");
        boton1.setBounds(10, 80, 100, 30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boton1) {
            String cad1 = textfield1.getText();
            String cad2 = textfield2.getText();
            try {
                int valor1 = Integer.parseInt(cad1);
                int valor2 = Integer.parseInt(cad2);
                int resultado = valor1 / valor2;
                setTitle("La división de " + cad1 + " con respecto a " + cad2 + " es " + resultado);
            } catch (NumberFormatException ex) {
                setTitle("Debe ingresar números enteros.");
            } catch (ArithmeticException ex) {
                setTitle("No se puede dividir por cero.");
            }
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1 = new Formulario();
        formulario1.setBounds(0, 0, 450, 170);
        formulario1.setDefaultCloseOperation(EXIT_ON_CLOSE);
        formulario1.setVisible(true);
    }
}
```

Cuando se presiona el botón en el método actionPerformed recuperamos los dos datos ingresados en los controles JTextField:

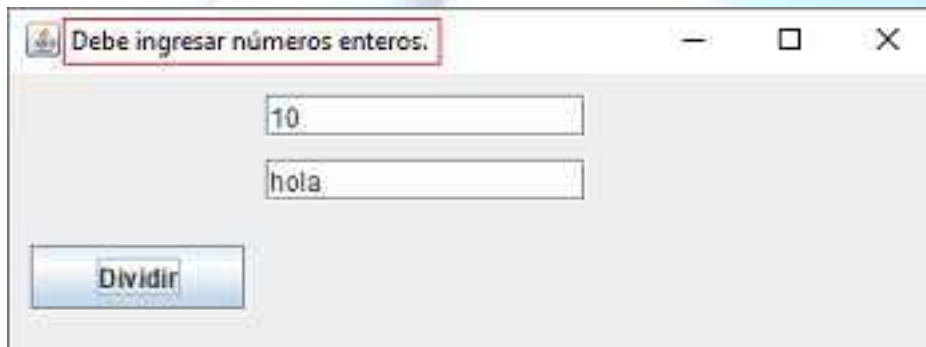
```
String cad1 = textfield1.getText();
String cad2 = textfield2.getText();
```

Abrimos un bloque try donde verificaremos excepciones de tipo 'NumberFormatException' que son disparadas por el método estático 'parseInt' de la clase Integer. Esto sucede si el String no contiene un valor entero.

De forma similar al problema anterior capturamos si hay una excepción de intentar la división por cero:

```
try {
    int valor1 = Integer.parseInt(cad1);
    int valor2 = Integer.parseInt(cad2);
    int resultado = valor1 / valor2;
    setTitle("La división de " + cad1 + " con respecto a " +
        cad2 + " es " + resultado);
} catch (NumberFormatException ex) {
    setTitle("Debe ingresar números enteros.");
} catch (ArithmeticException ex) {
    setTitle("No se puede dividir por cero.");
}
```

Si ejecutamos la aplicación e ingresamos en alguno de los dos controles de tipo JTextField valores que no sean numéricos, al presionar el botón se nos informa del problema en lugar de detenerse el programa con una excepción:



De forma similar se dispara la excepción ArithmeticException si ingresamos en el segundo JTextField el valor cero.

El bloque catch requiere en forma obligatoria las llaves de apertura y cerrado independientemente de que tenga 1 instrucción.

```
} catch (NumberFormatException ex) {
    setTitle("Debe ingresar números enteros.");
} catch (ArithmeticException ex) {
    setTitle("No se puede dividir por cero.");
}
```

Problema:

Declarar un vector de 10 elementos enteros. Permitir que el usuario ingrese un subíndice del vector y nos muestre el contenido de dicha componente. Atrapar las excepciones de fuera de rango del vector y si ingresa un valor no entero.

Programa:

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class ConsultaVector {

    public static void main(String[] ar) {
        int[] vec = { 20, 45, 76, 81, 34, 567, 423, 6, 3, 5 };
```

```

Scanner teclado = new Scanner(System.in);
int indice;
try {
    System.out.print("Ingrese un valor entre 0 y 9:");
    indice = teclado.nextInt();
    System.out.print("En el vector se almacena en la posición " + indice + " el valor " + vec[indice]);
} catch (InputMismatchException ex) {
    System.out.println("Debe ingresar obligatoriamente número entero");
} catch (IndexOutOfBoundsException ex) {
    System.out.println("El valor debe estar entre 0 y 9");
}
}
}

```

La excepción 'IndexOutOfBoundsException' se dispara cuando intentamos acceder a una componente inexistente de un vector (por ejemplo ingresamos un 15 en la variable 'indice'):

```

System.out.print("En el vector se almacena en la posición " +
    indice + " el valor " + vec[indice]);

```

Excepciones - no verificadas y verificadas

Existen dos tipos de excepciones en Java:

No verificadas.
Verificadas.

Las excepciones no verificadas son aquellas que dejan al programador tomar la decisión de la conveniencia de atraparla o no.

Todas las excepciones vistas hasta ahora son 'no verificadas': InputMismatchException, ArithmeticException, NumberFormatException y IndexOutOfBoundsException. Existen en el API de Java muchas otras excepciones de este tipo.

Si queremos una lista completa de excepciones no verificadas en Java podemos visitar la documentación oficial [aquí](#). Todas estas clases heredan de la clase java.lang.RuntimeException

Excepciones verificadas.

Este nuevo tipo de excepciones tienen la característica que deben ser capturadas en forma obligatoria por nuestro programa, si no la capturamos no se compila nuestra aplicación.

Las excepciones que se definen como verificadas generalmente son errores que no son directamente de nuestro programa sino a errores que surgen por ejemplo al intentar conectarnos a una base de datos, abrir un archivo inexistente, problemas de conexión de red etc.

La captura y tratamiento de la excepción verificada es idéntica a las excepciones no verificadas, con la salvedad que no podemos omitirla.

Problema:

Crear un archivo de texto con dos líneas. Luego proceder a leer el contenido del archivo de texto y mostrarlo por pantalla.

Programa:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

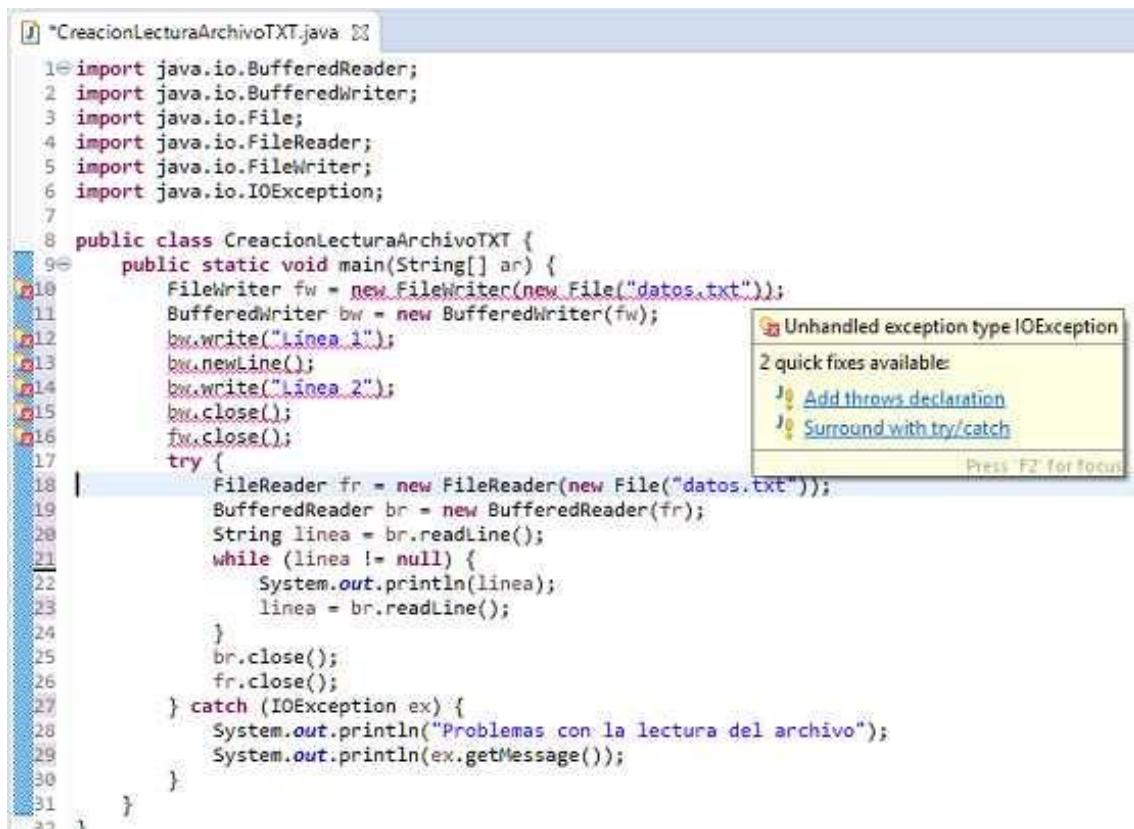
public class CreacionLecturaArchivoTXT {
    public static void main(String[] ar) {
        try {
            FileWriter fw = new FileWriter(new File("datos.txt"));
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("Línea 1");
            bw.newLine();
            bw.write("Línea 2");
            bw.close();
            fw.close();
        } catch (IOException ex) {
            System.out.println("Problemas en la creación del archivo");
            System.out.println(ex.getMessage());
        }
        try {
            FileReader fr = new FileReader(new File("datos.txt"));
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                System.out.println(linea);
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException ex) {
            System.out.println("Problemas con la lectura del archivo");
            System.out.println(ex.getMessage());
        }
    }
}
```

Para crear un archivo de texto utilizamos la clase `FileWriter`. Al constructor hay que pasar un objeto de la clase `File` indicando el nombre del archivo de texto a crear:

```
FileWriter fw = new FileWriter(new File("datos.txt"));
```

El constructor de la clase `FileWriter` si tiene un problema en la creación del archivo procede a elevar una excepción de tipo `IOException` que pertenece al grupo de excepciones 'verificadas'.

Podemos probar de no disponer el bloque `try/catch` y veremos que el compilador detecta que la clase `FileWriter` genera excepciones verificadas:



```
1 import java.io.BufferedReader;
2 import java.io.BufferedWriter;
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 public class CreacionLecturaArchivoTXT {
9     public static void main(String[] ar) {
10         FileWriter fw = new FileWriter(new File("datos.txt"));
11         BufferedWriter bw = new BufferedWriter(fw);
12         bw.write("Línea 1");
13         bw.newLine();
14         bw.write("Línea 2");
15         bw.close();
16         fw.close();
17         try {
18             FileReader fr = new FileReader(new File("datos.txt"));
19             BufferedReader br = new BufferedReader(fr);
20             String linea = br.readLine();
21             while (linea != null) {
22                 System.out.println(linea);
23                 linea = br.readLine();
24             }
25             br.close();
26             fr.close();
27         } catch (IOException ex) {
28             System.out.println("Problemas con la lectura del archivo");
29             System.out.println(ex.getMessage());
30         }
31     }
32 }
```

Como podemos comprobar el entorno de Eclipse nos informa que no hemos capturado la excepción.

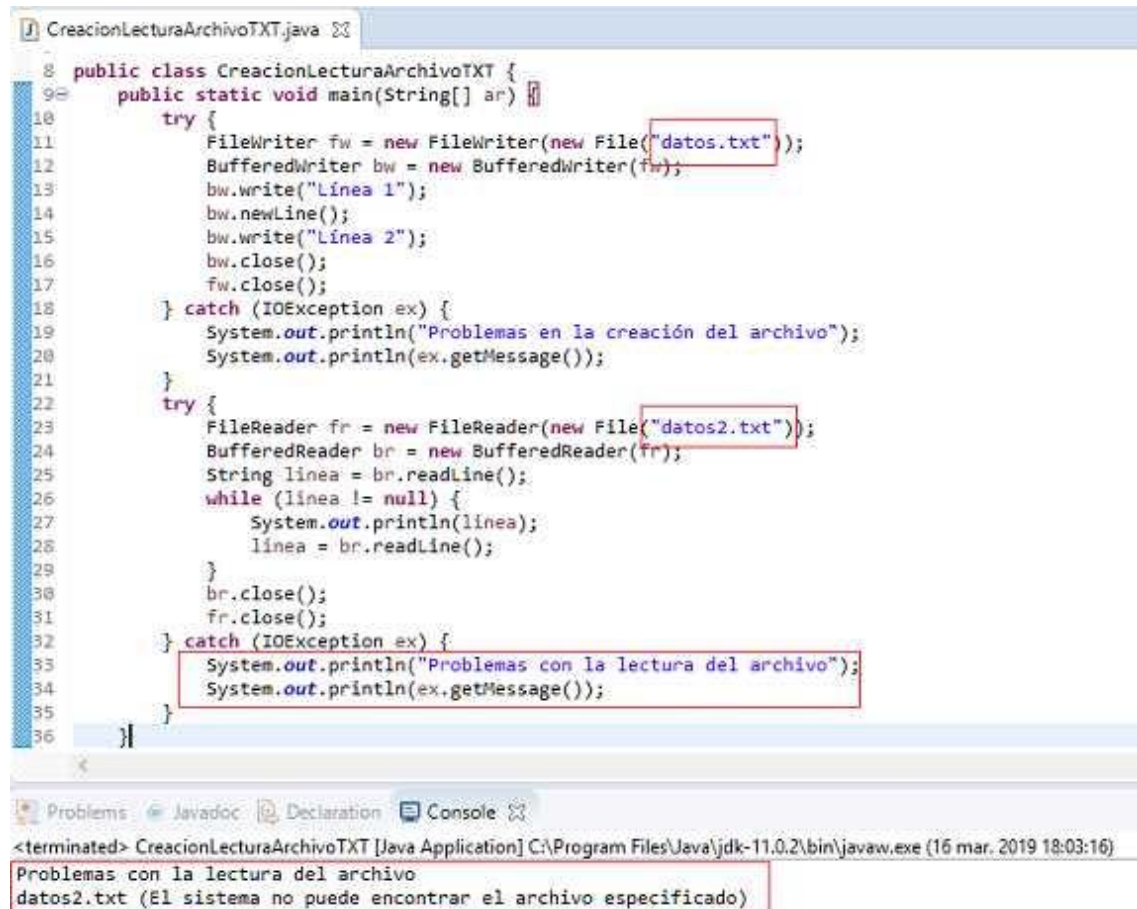
Utilizamos la clase BufferedWriter para hacer más eficiente la creación del archivo de texto:

```
try {
    FileWriter fw = new FileWriter(new File("datos.txt"));
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("Línea 1");
    bw.newLine();
    bw.write("Línea 2");
    bw.close();
    fw.close();
} catch (IOException ex) {
    System.out.println("Problemas en la creación del archivo");
    System.out.println(ex.getMessage());
}
```

La lectura del archivo de texto requiere la clase FileReader que también genera excepciones verificadas:

```
try {
    FileReader fr = new FileReader(new File("datos.txt"));
    BufferedReader br = new BufferedReader(fr);
    String linea = br.readLine();
    while (linea != null) {
        System.out.println(linea);
        linea = br.readLine();
    }
    br.close();
    fr.close();
} catch (IOException ex) {
    System.out.println("Problemas con la lectura del archivo");
    System.out.println(ex.getMessage());
}
```

Para ver que nos retorna el método `getMessage` de la clase `IOException` dispongamos la lectura de un archivo distinto al que creamos:



The screenshot shows an IDE with a Java file named `CreacionLecturaArchivoTXT.java`. The code defines a public class `CreacionLecturaArchivoTXT` with a `main` method. The `main` method has two try-catch blocks. The first block attempts to create and write to `datos.txt`, and the second block attempts to read from `datos2.txt`. Both blocks catch `IOException` and print error messages. The console output at the bottom shows the message: `Problemas con la lectura del archivo datos2.txt (El sistema no puede encontrar el archivo especificado)`.

```
8 public class CreacionLecturaArchivoTXT {
9     public static void main(String[] ar) {
10         try {
11             FileWriter fw = new FileWriter(new File("datos.txt"));
12             BufferedWriter bw = new BufferedWriter(fw);
13             bw.write("Línea 1");
14             bw.newLine();
15             bw.write("Línea 2");
16             bw.close();
17             fw.close();
18         } catch (IOException ex) {
19             System.out.println("Problemas en la creación del archivo");
20             System.out.println(ex.getMessage());
21         }
22         try {
23             FileReader fr = new FileReader(new File("datos2.txt"));
24             BufferedReader br = new BufferedReader(fr);
25             String linea = br.readLine();
26             while (linea != null) {
27                 System.out.println(linea);
28                 linea = br.readLine();
29             }
30             br.close();
31             fr.close();
32         } catch (IOException ex) {
33             System.out.println("Problemas con la lectura del archivo");
34             System.out.println(ex.getMessage());
35         }
36     }
37 }
```

Problems Javadoc Declaration Console

<terminated> CreacionLecturaArchivoTXT [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 mar. 2019 18:03:16)

Problemas con la lectura del archivo
datos2.txt (El sistema no puede encontrar el archivo especificado)

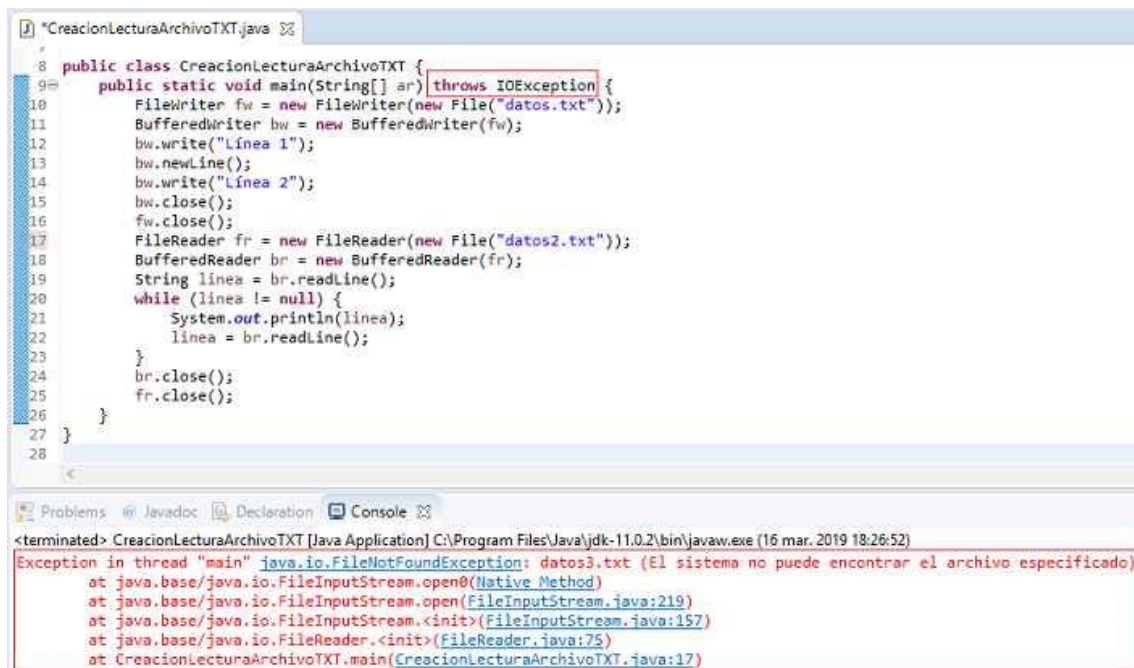
Todas estas clases heredan de la clase `java.lang.Exception` pertenecen a excepciones verificadas y debemos capturarlas.

Otras excepciones verificadas que podemos nombrar son: `SQLException`, `InterruptedException` etc.

Si queremos una lista completa de excepciones verificadas en Java podemos visitar la documentación oficial [aquí](#). Todas estas clases heredan de la clase `java.lang.Exception`

Comando `throws`

Podemos dejar que la máquina virtual de Java se encargue de las excepciones no verificadas agregando la palabra clave `throws` y el nombre de la excepción en el método `main` en este caso, luego si ocurre un error se detiene el programa y nos informa la excepción lanzada:

The screenshot shows an IDE window with a Java file named "CreacionLecturaArchivoTXT.java". The code defines a class with a main method that writes two lines to "datos.txt" and then attempts to read from "datos2.txt". The console output shows a "FileNotFoundException" for "datos2.txt".

```
8 public class CreacionLecturaArchivoTXT {
9     public static void main(String[] ar) throws IOException {
10         FileWriter fw = new FileWriter(new File("datos.txt"));
11         BufferedWriter bw = new BufferedWriter(fw);
12         bw.write("Línea 1");
13         bw.newLine();
14         bw.write("Línea 2");
15         bw.close();
16         fw.close();
17         FileReader fr = new FileReader(new File("datos2.txt"));
18         BufferedReader br = new BufferedReader(fr);
19         String linea = br.readLine();
20         while (linea != null) {
21             System.out.println(linea);
22             linea = br.readLine();
23         }
24         br.close();
25         fr.close();
26     }
27 }
28
```

Problems | Javadoc | Declaration | Console

<terminated> CreacionLecturaArchivoTXT [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (16 mar. 2019 18:26:52)

Exception in thread "main" java.io.FileNotFoundException: datos2.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 CreacionLecturaArchivoTXT.main(CreacionLecturaArchivoTXT.java:17)

Programa:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

```
public class CreacionLecturaArchivoTXT {
    public static void main(String[] ar) throws IOException {
        FileWriter fw = new FileWriter(new File("datos.txt"));
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write("Línea 1");
        bw.newLine();
        bw.write("Línea 2");
        bw.close();
        fw.close();
        FileReader fr = new FileReader(new File("datos2.txt"));
        BufferedReader br = new BufferedReader(fr);
        String linea = br.readLine();
        while (linea != null) {
            System.out.println(linea);
            linea = br.readLine();
        }
        br.close();
        fr.close();
    }
}
```

Tengamos en cuenta que hemos ingresado un error cuando tratamos de leer el archivo de texto 'datos2.txt', dicho archivo no existe.

Si utilizamos el comando throws en otro método distinto a la main luego quien llame a dicho método deberá implementar la excepción verificada o también propagar la excepción con un nuevo comando throws:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CreacionLecturaArchivoTXT {

    public static void crear() throws IOException {
        FileWriter fw = new FileWriter(new File("datos.txt"));
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write("Línea 1");
        bw.newLine();
        bw.write("Línea 2");
        bw.close();
        fw.close();
    }

    public static void leer() throws IOException {
        FileReader fr = new FileReader(new File("datos.txt"));
        BufferedReader br = new BufferedReader(fr);
        String linea = br.readLine();
        while (linea != null) {
            System.out.println(linea);
            linea = br.readLine();
        }
        br.close();
        fr.close();
    }

    public static void main(String[] ar) throws IOException {
        crear();
        leer();
    }
}
```

Los tres métodos propagan las excepciones mediante el comando throws.

La captura de excepciones verificadas y su tratamiento evitando el throws hace nuestros programas más robustos.

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar