



MANEJO DE ERRORES

Java Básico

¿Qué es una excepción?

2

- Las excepciones son las formas en que los programas de Java manejan las situaciones atípicas. Puede pensarse en una excepción como en un “error”.
- Cuando una excepción ocurre decimos que fue “lanzada”, y cuando manejamos dicha excepción, es decir hacemos algo al respecto del error, decimos que fue “capturada”
- Ejemplos de excepciones son:
 - Tratar de convertir la cadena “1 23A4” a número
 - Tratar de llamar a un método no estático de una referencia *null*

- ❑ Crea un proyecto llamado Errores.

```
public class Errores {  
    public static void main(String[] args) {  
        int numero = new Integer ("Hola");  
        System.out.println (numero);  
    }  
}
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "Hola"  
|   at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.lang.Integer.parseInt(Integer.java:492)  
    at java.lang.Integer.<init>(Integer.java:677)  
    at errores.Errores.main(Errores.java:17)
```

Java Result: 1

GENERACIÓN CORRECTA (total time: 0 seconds)

¿Qué es una excepción?

4

- Para capturar excepciones en un bloque de código debemos usar la instrucción **try..catch**.
- Por ejemplo:

```
try {  
    // sentencias a monitorear el error  
}  
catch (tipoexcepcion nombrevr) { ← Una o más  
    // sentencias de manejo de la excepción  
}  
catch (tipoexcepcion nombrevr) {  
    // sentencias de manejo de la excepción  
}  
finally { ← Opcional  
    //sentencias a ejecutar ocurran o no excepciones  
}
```

¿Qué es una excepción?

5

- Por ejemplo:

```
public static void main(String[] args) {  
    try{  
        int numero = new Integer ("Hola");  
        System.out.println (numero);  
    }  
    catch (NumberFormatException e){  
        System.out.println ("Error, la cadena no se puede convertir: ");  
    }  
}
```

¿Qué es una excepción?

6

- Agregamos una línea

```
public static void main(String[] args) {  
    try {  
        int numero = new Integer("Hola");  
        System.out.println(numero);  
    } catch (NumberFormatException e) {  
        System.out.println("Error, la cadena no se puede convertir: ");  
        e.printStackTrace();  
    }  
}
```

□ Crea otra clase llamada Errores2

```
public class Errores2 {  
    public static void main(String[] args) {  
        try {  
            String input =  
                JOptionPane.showInputDialog("Digite un número:");  
            int i = Integer.parseInt(input.substring(0));  
            System.out.println("El número es " + i);  
  
        } catch (NumberFormatException nfe) {  
            System.out.println("El formato del número es erroneo");  
        } catch (NullPointerException npe) {  
            System.out.println("Usted no ha digitado ningún número");  
        }  
    }  
}
```

Excepciones no tratadas

8

- Cuando en un programa se arroja una excepción y esta no es capturada, la excepción supera los límites del programa y es capturada por la JVM, mostrando un mensaje parecido a este:

```
Exception in thread "main"  
    java.lang.NullPointerException  
        at MiClase.main(MiClase.java:17)
```


¿Cómo arrojar una excepción?

9

- En ocasiones no solo debemos capturar excepciones predefinidas, sino que debemos crear nuestras propias excepciones y arrojarlas.
- Para arrojar una excepción debe usarse la palabra reservada ***throw***, que funciona se usa así:

```
...  
    if (elNumeroNoMeGustó)  
        throw new NumberFormatException()  
...
```

Tipos de excepciones

10

- Pueden distinguirse dos tipos de excepciones:
 - **Runtime Exceptions:** Son excepciones que se producen en el sistema de ejecución de Java. Tal como usar referencias null, hacer una división entre cero, acceder a un elemento inexistente en un array.
 - **NonRuntime Exceptions:** Son excepciones que se producen fuera del sistema de ejecución de Java. Son ejemplo de estas las excepciones que se producen por acceso a archivos (IOExceptions)
- En el segundo tipo de excepciones el compilador se asegura de que el programador maneje la excepción (es decir, que cree un bloque *try...catch*)

Tipos de excepciones

11

- La manera de distinguir ambos tipos de excepciones es mediante la clase de las que estas extienden (si, todas las excepciones son clases).
- Las excepciones del tipo Runtime deben extender de la clase *RuntimeException*, mientras las de tipo NonRuntime deben extender de *Exception*.

Excepciones NonRuntime

12

- Si tratáramos de compilar una clase que tuviera el siguiente método:

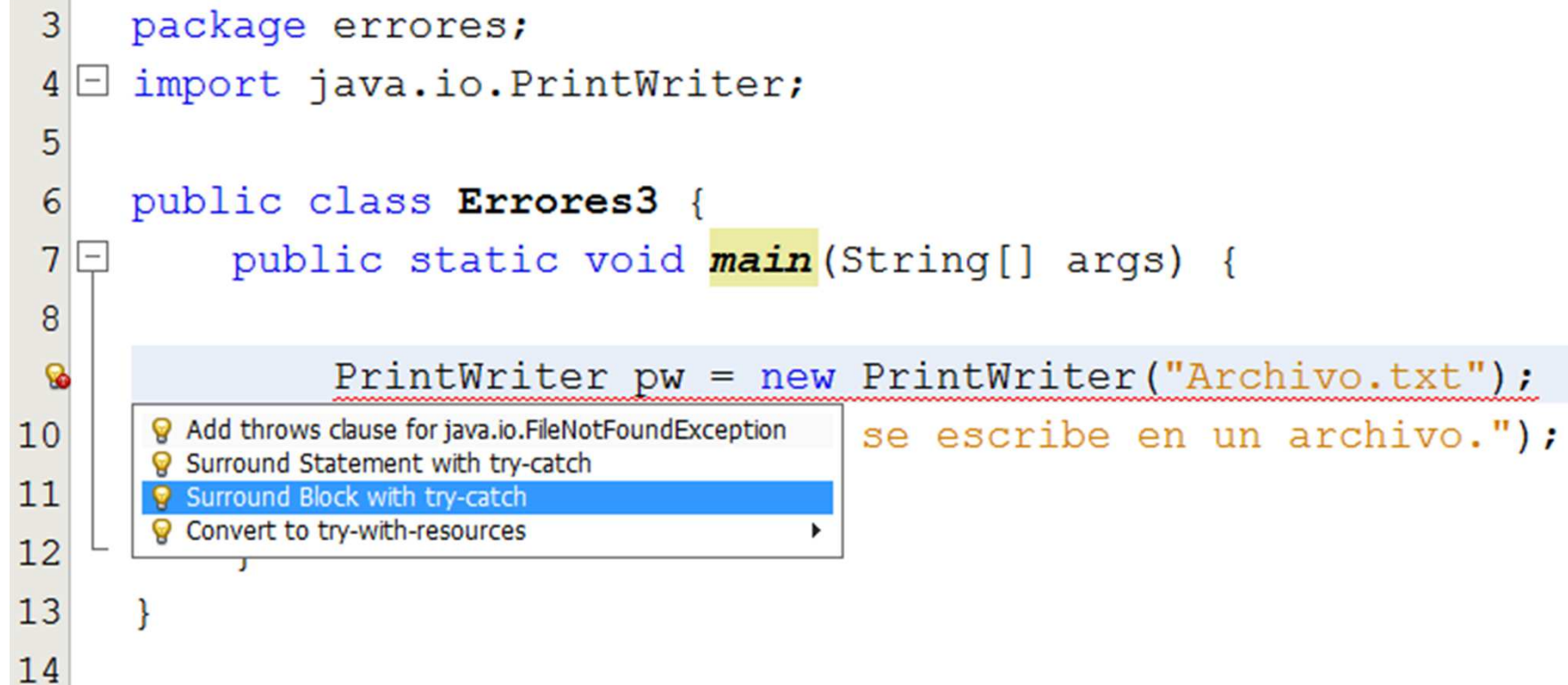
```
public class Errores3 {  
    public static void main(String[] args) {  
  
        PrintWriter pw = new PrintWriter("Archivo.txt");  
        pw.write("Este texto se escribe en un archivo.");  
        pw.close ();  
    }  
}
```

- El compilador nos daría este error

```
unreported exception java.io.FileNotFoundException;  
must be caught or declared to be thrown
```

- Debido a que no estamos capturando la excepción `FileNotFoundException` que puede ser lanzada

```
3 package errores;
4 import java.io.PrintWriter;
5
6 public class Errores3 {
7     public static void main(String[] args) {
8
9         PrintWriter pw = new PrintWriter("Archivo.txt");
10         se escribe en un archivo.";
11
12     }
13 }
14
```



The image shows a NetBeans IDE window with a Java file named `Errores3.java`. The code is as follows:

```
package errores;
import java.io.PrintWriter;

public class Errores3 {
    public static void main(String[] args) {

        PrintWriter pw = new PrintWriter("Archivo.txt");
        se escribe en un archivo.";

    }
}
```

A suggestion menu is open over the underlined line `PrintWriter pw = new PrintWriter("Archivo.txt");`. The menu contains the following options:

- ⚡ Add throws clause for java.io.FileNotFoundException
- ⚡ Surround Statement with try-catch
- ⚡ Surround Block with try-catch (highlighted)
- ⚡ Convert to try-with-resources

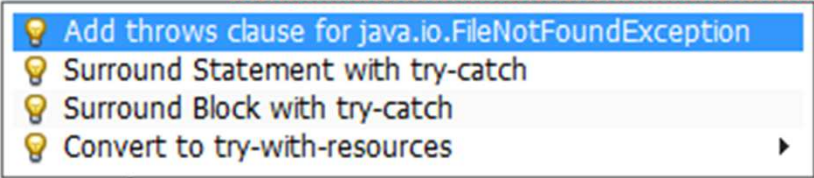
Excepciones NonRuntime

14

- Para librarnos del problema tenemos dos opciones:
 - Capturar la excepción:

```
public class Errores3 {  
    public static void main(String[] args) {  
        PrintWriter pw = null;  
        try {  
            pw = new PrintWriter("Archivo.txt");  
            pw.write("Este texto se escribe en un archivo.");  
            pw.close ();  
        } catch (FileNotFoundException ex) {  
            Logger.getLogger(Errores3.class.getName()).log(Lev  
        } finally {  
            pw.close();  
        }  
    }  
}
```

```
3 package errores;
4 import java.io.PrintWriter;
5
6 public class Errores3 {
7     public static void main(String[] args) {
8
9         PrintWriter pw = new PrintWriter("Archivo.txt");
10         se escribe en un archivo.");
11
12     }
13 }
14
```



The tooltip displays four suggestions for the error on line 10:

- Add throws clause for java.io.FileNotFoundException
- Surround Statement with try-catch
- Surround Block with try-catch
- Convert to try-with-resources

- O indicar en la declaración del método, que la excepción puede ser lanzada:

```
public static void main(String[] args) throws FileNotFoundException {  
  
    PrintWriter pw = new PrintWriter("Archivo.txt");  
    pw.write("Este texto se escribe en un archivo.");  
    pw.close ();  
}
```


Ejercicio: Captura las excepciones del siguiente código para que el programa no termine abruptamente.

17

```
public class EjercicioErrores {
    public static void main (String [] args){
        int [] contadores = new int [10];
        int n;

        do{
            String s = JOptionPane.showInputDialog (
                "Piensa un número (-1 para terminar) ");
            n = new Integer (s.substring(0));
            contadores[n]++;
        }while(n != -1);

        for(int i = 0;i<contadores.length; i++){
            System.out.println("Pensaste (" + i + ") " + contadores[i] + " veces");
        }
    }
}
```