Tratamiento de excepciones

Contenido

- El concepto de excepción.
- Captura y tratamiento de excepciones.
- Propagación de excepciones.
- Excepciones predefinidas.
- Definición de nuevas excepciones.

El concepto de “excepción”

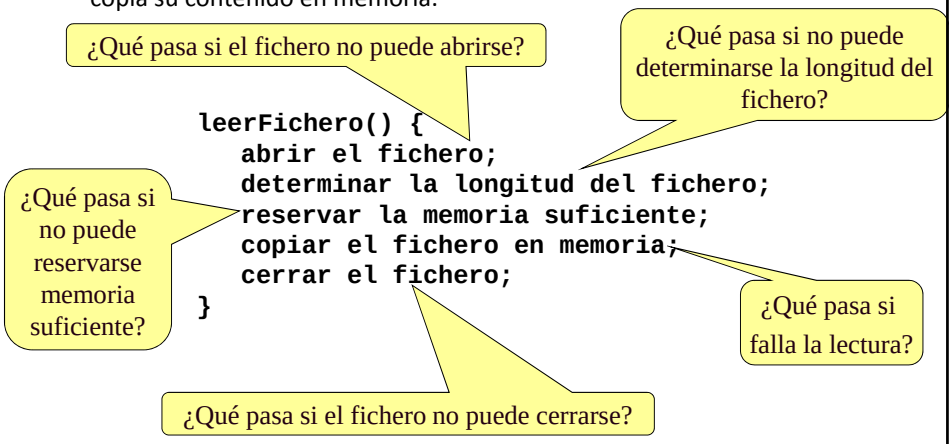
- Una *excepción* es un evento que interrumpe el flujo normal de instrucciones durante la ejecución del programa
- Durante la ejecución de un programa se pueden producir errores de diversos niveles de severidad:
  - un índice fuera de rango,
  - una división por cero.
  - un fichero que no puede encontrarse o no existe,
  - un enlace de red que falla, ...

El concepto de “excepción”

- El código se volvería ininteligible si continuamente tuviéramos que comprobar las posibles condiciones de error sentencia tras sentencia.

El concepto de "excepción"

- Consideremos el (pseudo)código del siguiente método que lee un fichero y copia su contenido en memoria.



El concepto de "excepción"

```
tipoDeCódigoDeError leerFichero() {
   tipoDeCódigoDeError códigoDeError = 0;
   abrir el fichero;
   if (el fichero está abierto) {
      determinar la longitud del fichero;
      if (se consigue la longitud del fichero) {
         reservar la memoria suficiente;
         if (se consigue la memoria) {
            copiar el fichero en memoria;
            if (falla la lectura) { códigoDeError = -1; }
         } else { códigoDeError = -2; }
      } else { códigoDeError = -3; }
      cerrar el fichero;
      if (códigoDeError == 0 && el fichero no se cerró){
         códigoDeError = -4;
      }
   } else { códigoDeError = -5; }
   return códigoDeE
}
```

- Difícil de leer
- Se pierde el flujo lógico de ejecución
- Difícil de modificar

El concepto de “excepción”

- Las excepciones son los mecanismos que Java proporciona para tratar de forma sistemática las situaciones excepcionales
- El mecanismo de las excepciones proporciona una forma clara de comprobar posibles errores sin oscurecer el código
  - Permite separar la lógica del programa de las situaciones excepcionales

El concepto de “excepción”

```
leerFichero() {
   try {
      abrir el fichero;
      determinar la longitud del fichero;
      reservar la memoria suficiente;
      copiar el fichero en memoria;
      cerrar el fichero;
   } catch (falló la apertura del fichero) {
      ...;
   } catch (falló el cálculo de la longitud del fichero) {
      ...;
   } catch (falló la reserva de memoria) {
      ...;
   } catch (falló la lectura del fichero) {
      ...;
   } catch (falló
      ...;
   }
}
```

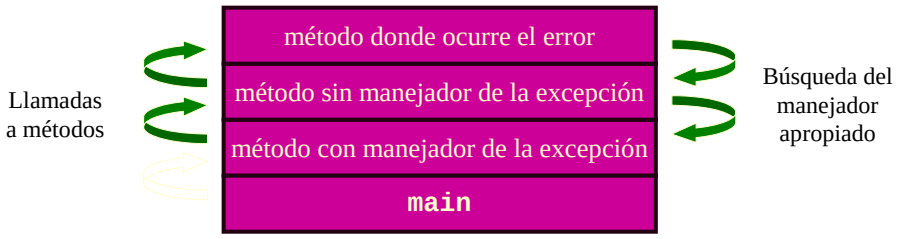
Las excepciones no nos liberan de hacer la detección, de informar y de manejar los errores, pero nos permiten escribir el flujo principal de nuestro código en un sitio y de tratar los casos excepcionales separadamente.

El concepto de "excepción"

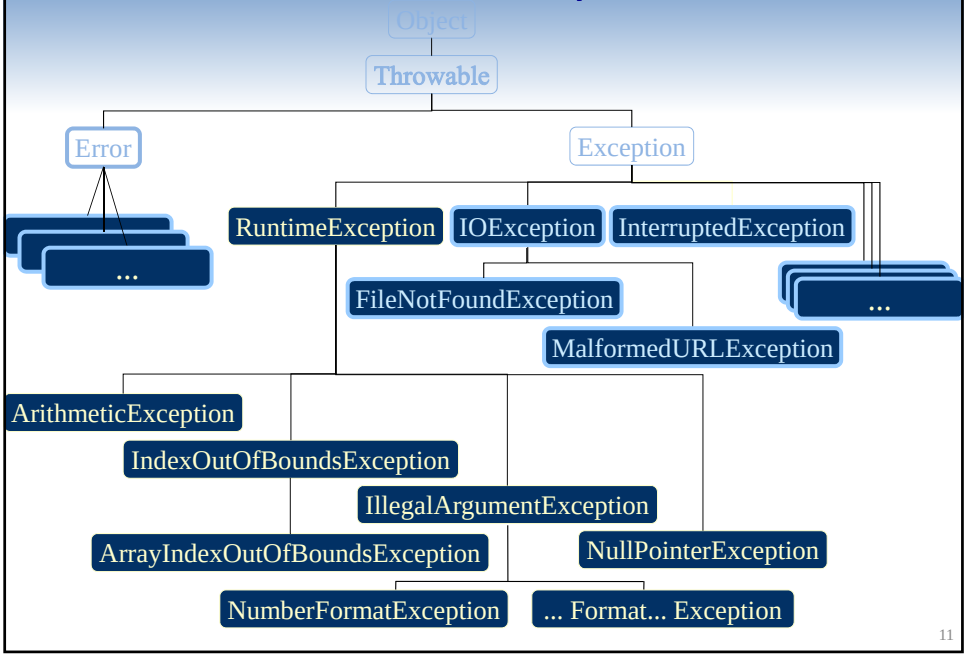
- Una *excepción* es un evento (una señal) que interrumpe el flujo normal de instrucciones durante la ejecución de un programa como consecuencia de un error (condición excepcional).
  - Decimos que se *lanza* una excepción (bien por el sistema o por el programa (throw)).
- La excepción puede ser *capturada* para su tratamiento (manejo).
- Si no es tratada, el programa termina (mostrando información detallada sobre ella).

El concepto de "excepción"

- Cuando ocurre un error en un método éste crea un objeto excepción (*una excepción*) y lo entrega al sistema de ejecución (*lanza una excepción*).
- Este objeto contiene información sobre el error, incluido su tipo y el estado del programa donde ocurrió.
- El sistema de ejecución recorre la "pila de llamadas" buscando un método que contenga un bloque de código que maneje la excepción (*manejador de excepción*).



La clase **`Throwable`** y sus subclases



La clase **`Throwable`**

- Sólo objetos que son instancias de la clase **`Throwable`** (o de una de sus subclases) pueden ser lanzados por la máquina virtual de Java o con una instrucción **`throw`**, y sólo éstos pueden ser argumento de una cláusula **`catch`**.
- Por convenio, la clase **`Throwable`** y sus subclases tienen dos constructores: uno sin argumentos y otro con un argumento de tipo **`String`**, el cual puede ser usado para producir mensajes de error.
- Un objeto de la clase **`Throwable`** contiene el estado de la pila de ejecución (de su *thread*) en el momento en que fue creado.

La clase `Throwable` (II)

Entre los distintos métodos que tiene están:

`String getMessage()`
Devuelve el texto con el mensaje de error del objeto.

`void printStackTrace()`
Imprime este objeto y su traza en la salida de errores estándar.

Lanzar una excepción

- Una excepción es una instancia de una clase que se crea con new.
  - La instancia de clase que se crea, debe ser de la clase **`Throwable`** o de una de sus subclases
- Para lanzarla se utiliza

```
throw excepción
```

- Esto interrumpe el flujo de ejecución y se procede a la búsqueda de un manejador para esa excepción, es decir, alguno que la trate.
- Además de lanzar las excepciones predefinidas, es posible definir y lanzar nuevas excepciones
  - El tratamiento es el mismo que para la excepciones prefefinidas lanzadas por el sistema

Lanzar una excepción

- Una excepción es una instancia de una clase, que se crea con el operador new.
- Para lanzarla se utiliza
  `throw <objeto-excepción>`
  `throw new RuntimeException("comentario adecuado");`
- Esto interrumpe el flujo de ejecución y se procede a la búsqueda de un manejador para esa excepción, es decir, "alguien que la trate".

```
public Circulo(Punto p, double r) {
    if (r < 0 ) {
        throw new RuntimeException("Radio negativo");
    }
    radio = r;
    centro = p;
}
```


Lanzar una excepción

En la clase **`Recta`** del proyecto **`prRecta`**

```
public Punto interseccionCon(Recta r){
    if (paralelaA(r)){
        throw new RuntimeException("Rectas paralelas");
    }
    ...
}
```

¡Nos piden que devolvamos un punto pero son paralelas!

No sabemos qué hacer. Lanzamos la excepción

En la clase **`Urna`** del proyecto **`prUrna`**

```
public class Urna {
    ...
    public ColorBola extraerBola() {
        if (totalBolas()==0) {
            thrown new RuntimeException("No hay bolas");
        }
        ...
    }
}
```

¡Nos piden una bola y no hay!

No sabemos qué hacer. Lanzamos la excepción

Propagar una excepción

- Una excepción será automáticamente propagada si no se trata

```
public int stringAInt(string str){
    int n = Integer.parseInt(str);
    return n;
}
```

Si **`str`** no es convertible a entero nos lanzan una **`NumberFormatException`** y la propagamos

Propagar una excepción

- En la clase **`TestUrna`** del proyecto **`prUrna`**

```
public class TestUrna {
    public static void main(string [] args) {
        int bb = Integer.parseInt(args[0]);
        int bn = Integer.parseInt(args[1]);
        ...
    }
}
```

Si **`args[0]`** o **`args[1]`** no es convertible a entero nos lanzan una **`NumberFormatException`** y la propagamos

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "2e"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:458)
at java.lang.Integer.parseInt(Integer.java:499)
at TestUrna.main(TestUrna.java:5)
```

Captura de excepciones
La instrucción `try-catch`

- Vigila bloques de código para capturar las posibles excepciones.
- Las instrucciones a vigilar se encierran en bloques **`try`**.
- Los manejadores de excepciones se incluyen en bloques **`catch`** asociados a éstos.

```
try {
  ...
} catch (TipoDeExcepción nombre) {
  ...
} catch (TipoDeExcepción nombre) {
  ...
} ...
```

- Las instrucciones de un bloque **`catch`** son ejecutadas cuando se invoca dicho manejador de excepciones, es decir, cuando el bloque vigilado lanza la excepción.

Captura de excepciones
La instrucción `try-catch`

```
try {
  ...
} catch (TipoDeExcepción nombre) {
  ...
} catch (TipoDeExcepción nombre) {
  ...
} ...
```

- Cada bloque **`catch`** proporciona un manejador de excepción que trata el tipo de manejador indicado en su único argumento, **`TipoDeExcepción,`** que declara el tipo de excepción que el manejador puede trata (clase **`Throwable)`**.
- Las sentencias del bloque serán invocadas cuando el manejador sea invocado.
- El sistema de ejecución invoca al manejador, cuando este es el primero en la pila de ejecución, cuyo tipo **`TipoDeExcepción`**, coincide con el tipo de la excepción lanzada

La captura de excepciones

```
public class TestUrna {
    public static void main(string [] args) {
      try {
            int bb = Integer.parseInt(args[0]);
            int bn = Integer.parseInt(args[1]);
            ...
      } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Uso: TestUrna <Int> <Int>");
      } catch (NumberFormatException ee) {
            System.out.println("Los args deben ser enteros");
      }
    }
}
```

- Los manejadores pueden hacer más: preguntar al usuario por la decisión a tomar, recuperarse del error o terminar el programa.
- También podemos poner cada una de las instrucciones que pueden lanzar excepciones en bloques **`try`** diferentes y proporcionar manejadores de excepciones para cada uno.

El bloque **`finally`**

- El bloque **`finally`** es opcional, y su función es la de dejar el programa en un estado correcto independientemente de lo que suceda dentro del bloque **`try`** (cerrar ficheros, liberar recursos, ...).

- El bloque **`finally`** es ejecutado siempre.

Ejemplo: uso del bloque **`finally`**

```
public void escribeLista() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("out.txt"));
            for (int i = 0; i < SIZE; i++) {
                out.println("valor: " + i + " = " + v[i]);
     }
    } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Índice fuera de rango");
    } catch (IOException e) {
            System.out.println("out.txt no puede abrirse");
    } finally {
            if (out != null) {
                out.close();
            }
    }
}
```


1: Ocurre una excepción **`IOException`**

```
public void escribeLista() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("out.txt"));
            for (int i = 0; i < SIZE; i++) {
                out.println("valor: " + i + " = " + v[i]);
     }
    } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Índice fuera de rango");
    } catch (IOException e) {
            System.out.println("out.txt no puede abrirse");
    } finally {
            if (out != null) {
                out.close();
            }
    }
}
```

2: Ocurre una excepción `ArrayIndexOutOfBoundsException`

```
public void escribeLista() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("out.txt"));
            for (int i = 0; i < SIZE; i++) {
                out.println("valor: " + i + " = " + v[i]);
     }
    } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Índice fuera de rango");
    } catch (IOException e) {
            System.out.println("out.txt no puede abrirse");
    } finally {
            if (out != null) {
                out.close();
            }
    }
}
```


3: El bloque `try` termina normalmente

```
public void escribeLista() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("out.txt"));
            for (int i = 0; i < SIZE; i++) {
                out.println("valor: " + i + " = " + v[i]);
     }
    } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Índice fuera de rango");
    } catch (IOException e) {
            System.out.println("out.txt no puede abrirse");
    } finally {
            if (out != null) {
                out.close();
            }
    }
}
```

Ejemplo 2: uso de finally

```
public class DPC {
    public void división(int num1, int num2) {
            System.out.println(num1 + " / " + num2 + " = " + (num1 / num2));
            System.out.println("Volviendo de división.");
    }
    public static void main(String[] args) {
      new DPC().división(10, 0);
      System.out.println("Volviendo de main.");
    }
}
```

Salida:

```
java.lang.ArithmeticException: / by zero
at DPC.división(DPC.java:4)
at DPC.main(DPC.java:9)
Exception in thread "main"
```


Ejemplo 3: uso de **Finally**

```
public class DPC {
    public void división(int num1, int num2) {
      try {
            System.out.println(num1 + " / " + num2 + " = " + (num1 / num2));
      } finally {
            System.out.println("Finally hecho.");
      }
      System.out.println("Volviendo de división.");
    }
    public static void main(String[] args) {
      new DPC().división(10, 0);
      System.out.println("Volviendo de main.");
    }
}
```

Salida:

```
Finally hecho.
java.lang.ArithmeticException: / by zero
at DPC.división(DPC.java:5)
at DPC.main(DPC.java:12)
Exception in thread "main"
```

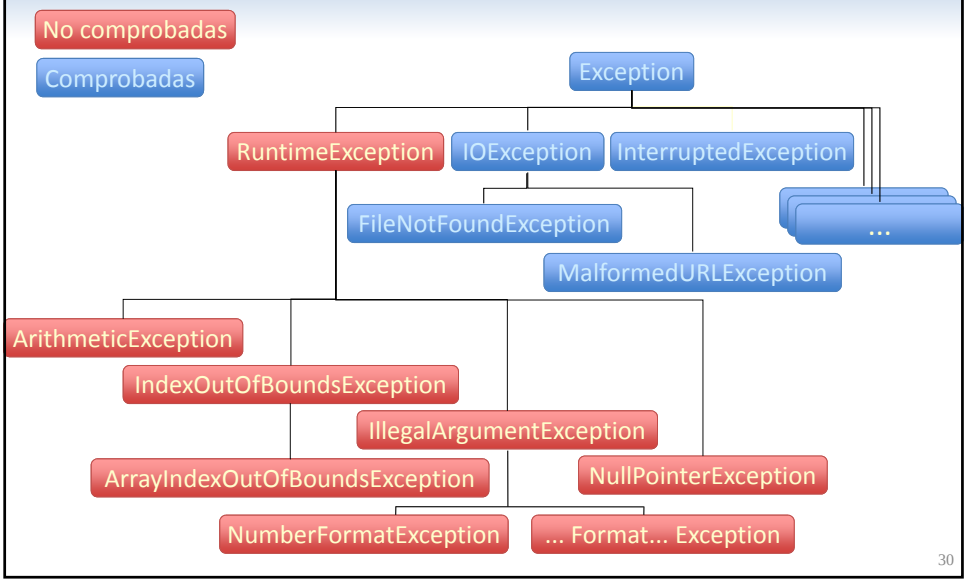
Ejemplo 4: uso de **Finally**

```
public class DPC {
    public void división(int num1, int num2) {
      try {
            System.out.println(num1 + " / " + num2 + " = " + (num1 / num2));
      } finally {
            System.out.println("Finally hecho.");
      }
      System.out.println("Volviendo de división.");
    }
    public static void main(String[] args) {
      try {
            new DPC().división(10, 0);
      } catch (ArithmeticException e) {
            System.out.println("División por cero.");
      }
      System.out.println("Volviendo de main.");
    }
}
```

Salida:

```
Finally hecho.
División por cero.
Volviendo de main.
```


Las excepciones de obligado tratamiento (checked o comprobadas)



Excepciones comprobadas

- Las excepciones comprobadas deben ser tratadas.

```
public void escribeListaHasta() {
      PrintWriter out = null;
      try {
                  out = new PrintWriter(new FileWriter("out.txt"));
                  for (int i = 0; i < SIZE; i++) {
                        out.println("valor: " + i + " = " + v[i]);
                  }
      } catch (IOException e) {
                  System.out.println("out.txt no puede abrirse");
      } finally {
                  if (out != null) {
                        out.close();
                  }
      }
}
```


Tratamiento de excepciones comprobadas

- Las excepciones comprobadas deben ser capturadas o anunciadas
- Capturadas: se hace un tratamiento con ellas.
- Anunciadas o propagadas: se anuncia en la cabecera del método (aportan información al usuario del método; tan importantes como el tipo de los parámetros y el tipo devuelto)

```
public void escribeLista() throws IOException {
      PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
      for (int i = 0; i < SIZE; i++) {
            out.println("valor: " + i + " = " + v[i]);
      }
      out.close();
}
```

- Puede anunciarse más de una excepción separadas por comas
- Las excepciones no comprobadas (ej. ArrayIndexOutOfBounds) si queremos las podemos anunciar también.

Excepciones Comprobadas

```
import java.io.*;

public class ListaDeNúmeros {
    private int[] v;
    private static final int SIZE = 10;

    public ListaDeNúmeros() {
            v = new int[SIZE];
            for (int i = 0; i < SIZE; i++) {
                    v[i] = i;
            }
    }

    public void escribeLista() {
            PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
            for (int i = 0; i < SIZE; i++) {
                    out.println("valor: " + i + " = " + v[i]);
            }
            out.close();
    }
}
```

¡Este código no compila!

Las excepciones comprobadas (checked exceptions), como `IOException`, han de ser tratadas obligatoriamente en el programa.

Si el fichero no puede abrirse lanza una excepción `IOException`

Redefinición de métodos con cláusula `throws`

La definición del método en la subclase puede especificar todas o un subconjunto de las clases de excepciones (incluidas sus subclases) especificadas en la cláusula `throws` del método redefinido en la superclase.

```
class A {
    ...
    protected void métodoX()
    throws Excepción1, Excepción2, Excepción3 {
        ...
    }
    ...
}
class B extends A {
    ...
    protected void métodoX()
    throws Excepción1, Subc1Excepción3, subc2Excepción3 {
        ...
    }
    ...
}
```

Excepciones relacionadas

```
public void escribeLista() {
    try {
     PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
     for (int i = 0; i < SIZE; i++) {
          out.println("valor: " + i + " = " + v[i]);
     }
     out.close();
    } catch (IOException e) {
     System.out.println("Error de entrada/salida");
    } catch (FileNotFoundException e) {
     System.out.println("out.txt no puede abrirse");
    }
}
```

Excepciones relacionadas

```
public void escribeLista() {
    try {
     PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
     for (int i = 0; i < SIZE; i++) {
          out.println("valor: " + i + " = " + v[i]);
     }
     out.close();
    } catch (FileNotFoundException e) {
     System.out.println("out.txt no puede abrirse");
    } catch (IOException e) {
     System.out.println("Error de entrada/salida");
    }
}
```

Ejemplo: información sobre las excepciones

```
public class Ejemplo {
  void aux() {
     try {
        int a[] = new int[2];
        a[4] = 0;
     } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("excepción: " + e.getMessage());
        e.printStackTrace();
     }
  }
  public static void main(String[] args) {
     new Ejemplo().aux();
     System.out.println("fin");
  }
}
```

Salida:

```
Excepción 4
Fin
java.lang.ArrayIndexOutOfBoundsException 4
   at Ejemplo.aux(Ejemplo.java:5)
   at Ejemplo.main(Ejemplo.java:13)
```


Ejemplo 2: información sobre las excepciones

```
public class Ejemplo {
  void aux() {
        int a[] = new int[2];
        a[4] = 0;
  }
  public static void main(String[] args) {
     new Ejemplo().aux();
     System.out.println("fin");

  }
}
```

Puesto que la excepción no es captura, el programa y no imprime “fin”

Salida:

```
Exception in thread "main“
java.lang.ArrayIndexOutOfBoundsException: 4
at Ejemplo.aux(Ejemplo.java:6)
at Ejemplo.main(Ejemplo.java:13)
```

Definiendo nuestras propias excepciones

Un usuario puede definir sus propias excepciones.

```
public class MiExcepción extends Exception {
        public MiExcepción() {
                super();
        }
        public MiExcepción(String msg) {
                super(msg);
        }
        public MiExcepción(int i) {
                super(Integer.toString(i));
        }
}
```

- Y ahora puede lanzarse como las demás.

```
throw new MiExcepción(5);
```


Ejemplo: lanzando nuestra excepción

```
public class Ejemplo {
    public void división(int num1, int num2) {
      if (num2 == 0) {
            throw new MiExcepcion(num1);
      }
      System.out.println(num1 + " / " + num2 + " = " + (num1 / num2));
    }
    public static void main(String[] args) {
      try {
            new Ejemplo().división(10, 0);
            System.out.println("División hecha.");
      } catch (MiExcepcion e) {
            System.out.println("Número " + e.getMessage());
      } finally {
            System.out.println("Finally hecho.");
      }
    }
}
```

```
Salida:
Número 10
Finally hecho.
```

Ejemplo 2: lanzando nuestra excepción

```
public class DivisiónPorCero {

    public void división(int num1, int num2) {

        try {
            if (num2 == 0) {
               throw new MiExcepción("/ por 0");
            }

            System.out.println(
              num1 + " / " + num2 + " = " + (num1 / num2));
        }
        catch (MiExcepción e) {
            System.out.println("Trata " + e);
        } finally {
            System.out.println("Finally hecho.");
        }
        System.out.println("Volviendo de división.");
    }
    public static void main(String[] args) {
        new DivisiónPorCero().división(10, 0);
        System.out.println("Volviendo de main.");
    }
}
```

Salida:

```
Trata MiExcepción: / por 0
Finally hecho.
Volviendo de división.
Volviendo de main.
```


Reglas para tratar situaciones excepcionales

- **Preventiva**:
  - La comprobación es poco costosa y es probable que se produzca la excepción.
  - Ejemplo:
    - El método **`interseccionCon(Recta r)`** de **`Recta`**
    - El método **`extraerBola()`** de **`Urna`**

- **Curativa**:
  - La comprobación es costosa y es raro que se produzca la excepción.
  - Ejemplo:

```
public int stringAInt(string str){
  int n = Integer.parseInt(str);
  return n;
}
```