

Tratamiento de excepciones

Contenido

- Sw. tolerante a fallos.
 - 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 de un programa.
- Las aplicaciones pueden producir muchas clases de errores de diversos niveles de severidad:
 - un fichero que no puede encontrarse o no existe,
 - un índice fuera de rango,
 - un enlace de red que falla,
 - un fallo en un disco duro,
 - ...

La necesidad de tratar los errores

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

¿Qué pasa si el fichero no puede abrirse?

¿Qué pasa si no puede determinarse la longitud del fichero?

¿Qué pasa si no puede reservarse memoria suficiente?

```
leerFichero() {  
    abrir el fichero;  
    determinar la longitud del fichero;  
    reservar la memoria suficiente;  
    copiar el fichero en memoria;  
    cerrar el fichero;  
}
```

¿Qué pasa si falla la lectura?

¿Qué pasa si el fichero no puede cerrarse?

Tratamiento clásico de errores

```
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 (el fichero no se cerró && códigoDeError == 0) {
            códigoDeError = -4;
        } else { códigoDeError = códigoDeError and -4; }
    } else { códigoDeError = -5; }
    return códigoDeError;
}
```

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

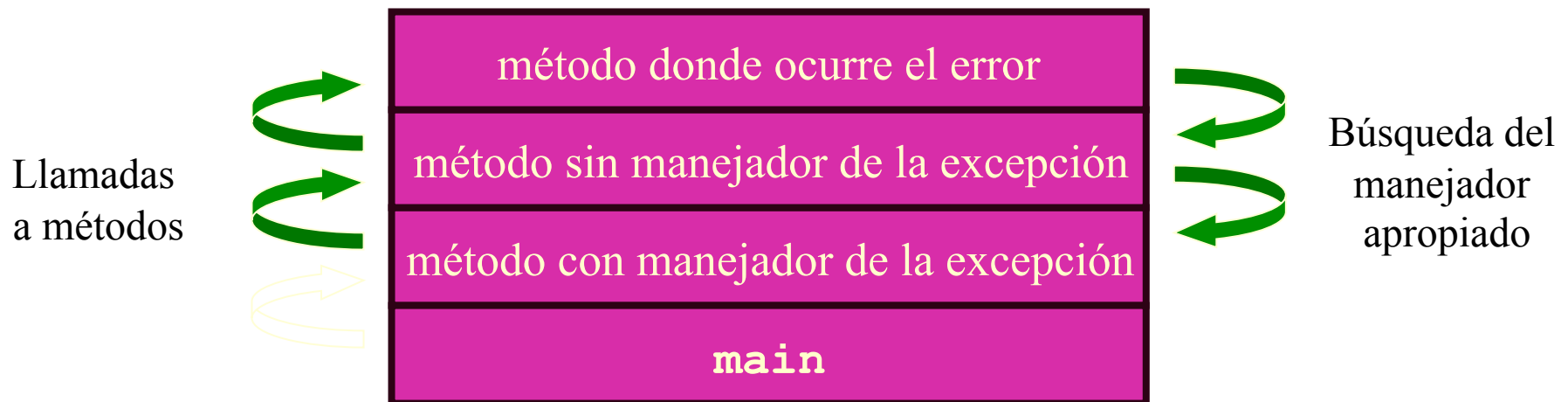
El tratamiento de excepciones

```
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.

¿Qué es una excepción?

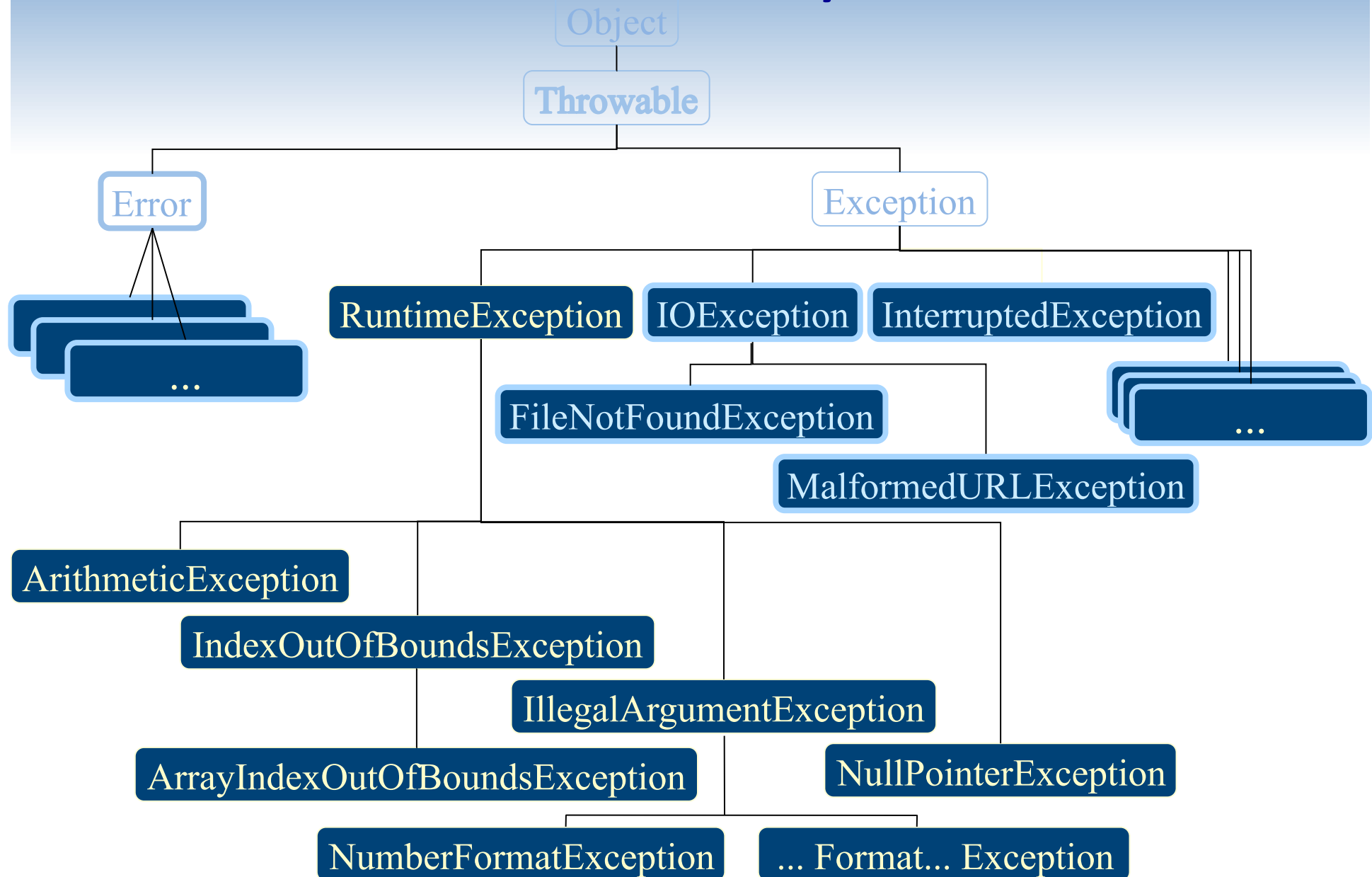
- Cuando ocurre un error en un método éste crea un objeto excepción (*una excepción*) y lo entrega al sistema de ejecución (*lanza una excepción*).
- Este objeto contiene información sobre el error, incluido su tipo y el estado del programa donde ocurrió.
- El sistema de ejecución recorre la pila de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



Puntos de vista ante la excepciones

- De lo visto hasta ahora hay dos puntos de vista para enfocar el tema de las excepciones
 - El que lanza (eleva o propaga) la excepción.
 - Le puede llegar la excepción y no saber que hacer con ella (la propaga).
 - Puede encontrarse en una situación que no sabe resolver y por tanto crear una excepción y lanzarla.
 - El que la trata (captura) la excepción
 - Sabe cómo resolver una situación provocada por una excepción y por tanto dispone de un tratamiento para ella.
- Un método puede actuar desde los dos puntos de vista
 - Captura unas excepciones y lanza otras.
- Las especificaciones del problema definen la mayoría de las veces cómo actuar.

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)

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.

void printStackTrace(PrintStream s)

Imprime este objeto y su traza en el canal especificado.

void printStackTrace(PrintWriter s)

Imprime este objeto y su traza en el *print writer* especificado.

Lanzar una excepción

- Una excepción es una instancia de una clase que se crea con new.
- 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, alguien que la trate.


```
throw new RuntimeException("comentario adecuado");
```

```
public void pop() {  
    if (size == 0) {  
        throw new EmptyStackException();  
    }  
    setObjectAt(size - 1, null);  
    size--;  
}
```

Lanzar una excepción

En la clase `Recta` del ejemplo `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) {  
            throw 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**

- Vigilan 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.

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.

El 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.err.println("Índice fuera de rango");  
    } catch (IOException e) {  
        System.err.println("out.txt no puede abrirse");  
    } finally {  
        if (out != null) {  
            out.close();  
        }  
    }  
}
```

1: Ocorre 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.err.println("Índice fuera de rango");  
        } catch (IOException e) {  
            System.err.println("out.txt no puede abrirse");  
        } finally {  
            if (out != null) {  
                out.close();  
            }  
        }  
    }  
}
```


2: Ocorre 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.err.println("Índice fuera de rango");  
    } catch (IOException e) {  
        System.err.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.err.println("Índice fuera de rango");  
    } catch (IOException e) {  
        System.err.println("out.txt no puede abrirse");  
    } finally {  
        if (out != null) {  
            out.close();  
        }  
    }  
}
```

A diagram consisting of two blue curly braces. The first brace is on the left side of the code, spanning from the 'try' block down to the first '}' of the 'finally' block. The second brace is on the left side of the code, spanning from the 'finally' block down to the final closing brace of the 'public void escribeLista()' method.

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"

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.");  
    }  
}
```

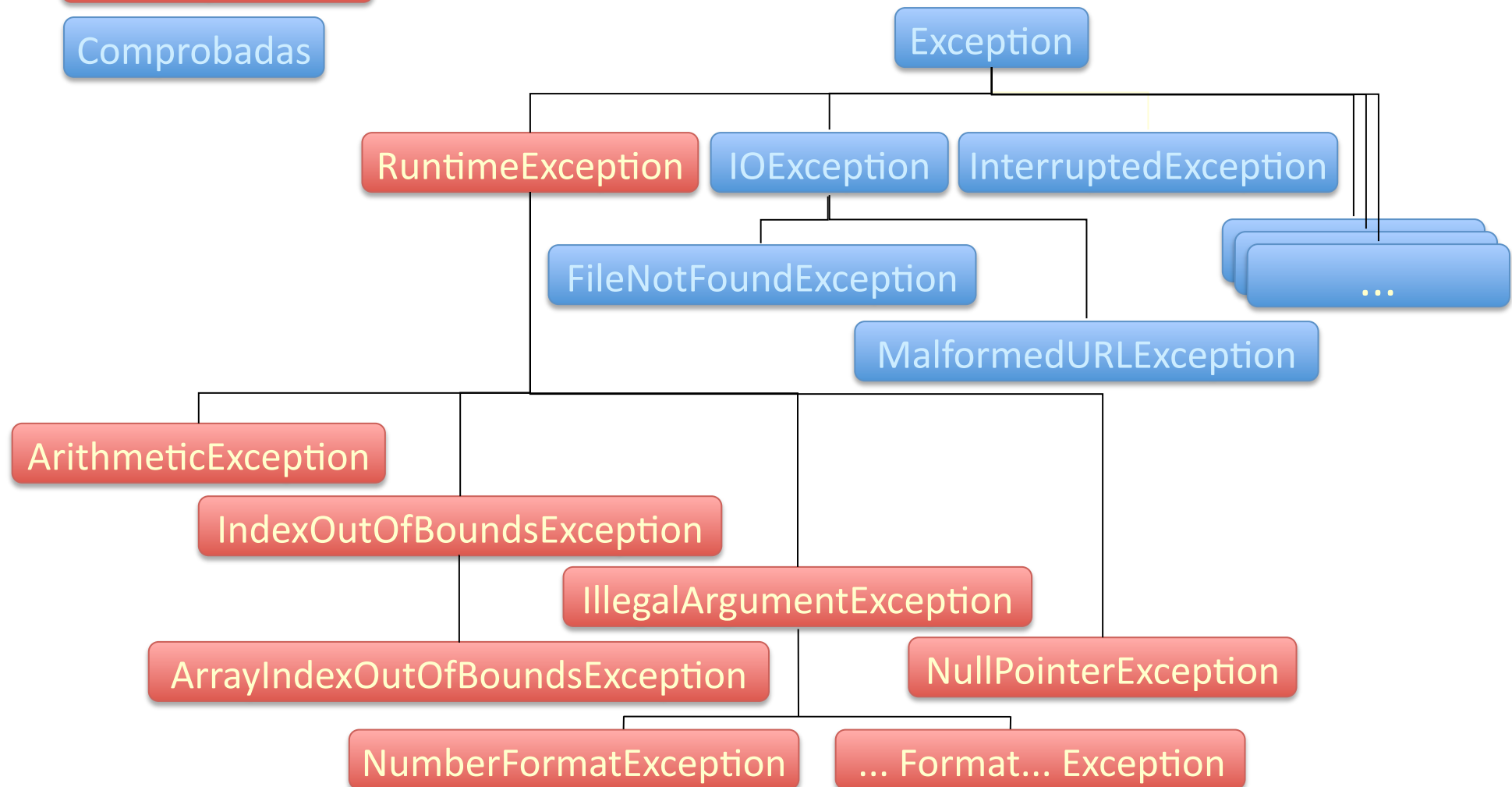
Salida:

Finally hecho.
División por cero.
Volviendo de main.

Las excepciones de obligado tratamiento (checked o comprobadas)

No comprobadas

Comprobadas



Tratamiento de excepciones comprobadas

- Las excepciones comprobadas deben ser capturadas o anunciadas
- Capturadas: se hace un tratamiento con ellas.
- Anunciadas: se anuncia en la cabecera del método.

```
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();  
}
```

- Pueden anunciarse varias excepciones, separadas por comas
- Las excepciones no comprobadas si queremos las podemos anunciar también.

Redefinición de métodos con cláusula **throws**

La definición del método en la subclase sólo puede especificar 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 (FileNotFoundException e) {  
        System.err.println("out.txt no puede abrirse");  
    } catch (IOException e) {  
        System.err.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  
java.lang.ArrayIndexOutOfBoundsException  
    at Ejemplo.aux(Ejemplo.java:5)  
    at Ejemplo.main(Ejemplo.java:13)  
fin
```

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) throws MiExcepcion {  
        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.

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;  
}
```