

Tratamiento de excepciones

Contenido

- Software 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?

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

¿Qué pasa si no puede reservarse memoria suficiente?

¿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

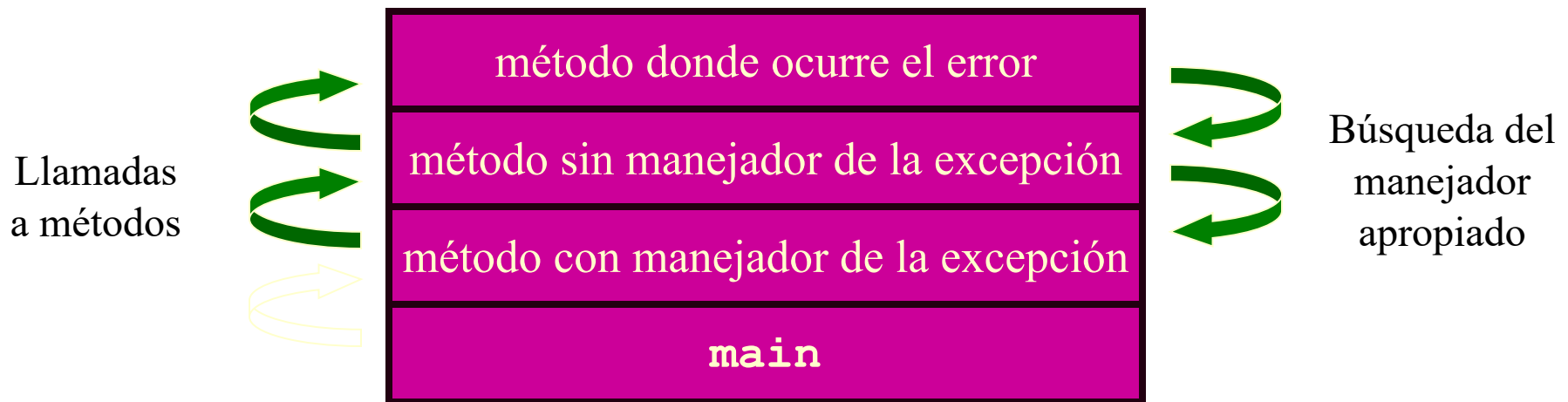
El tratamiento de excepciones

```
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?

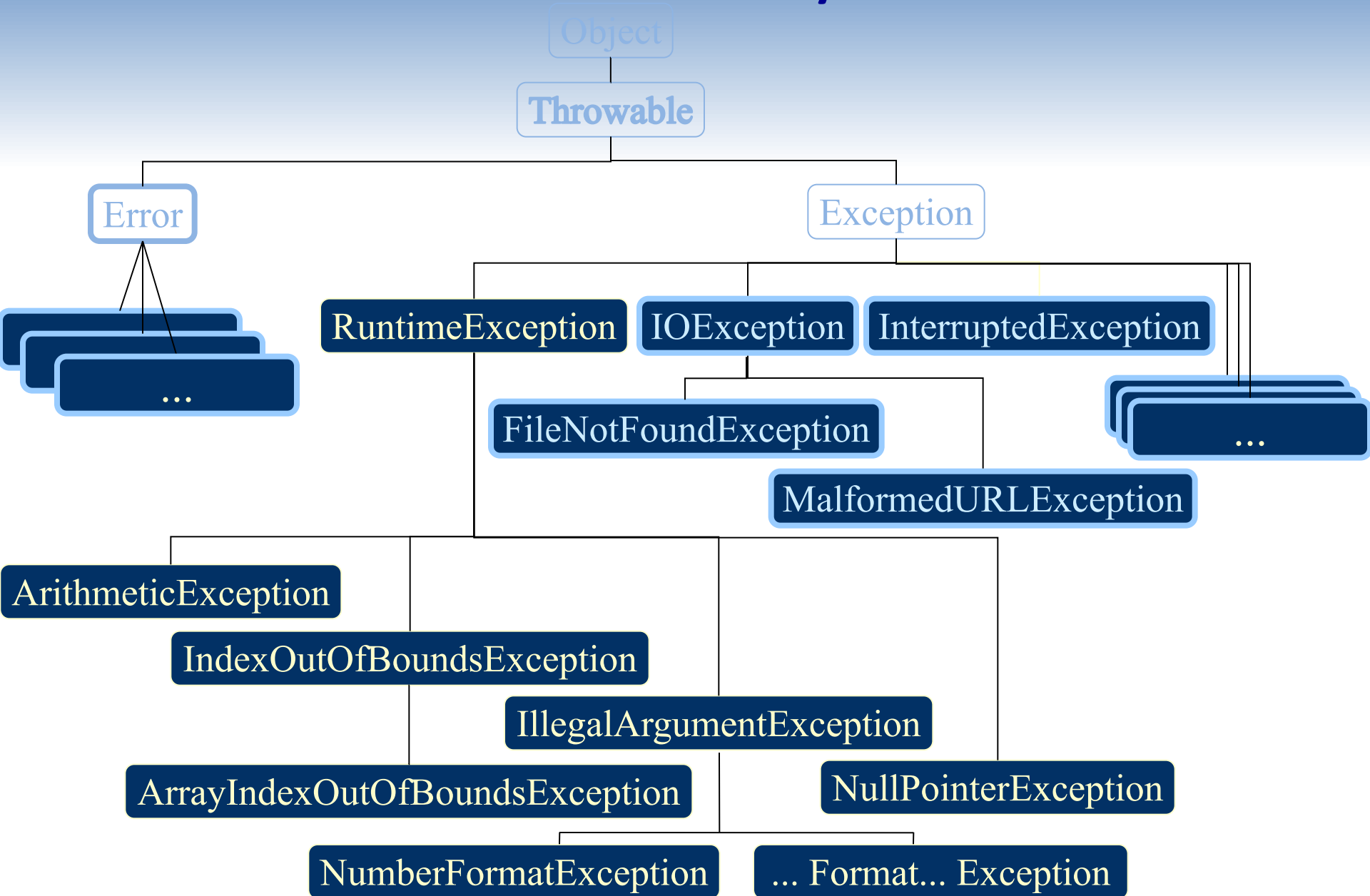
- Cuando ocurre un error en un método éste crea un objeto excepción (*una excepción*) y lo entrega al sistema de ejecución (*lanza una excepción*).
- Este objeto contiene información sobre el error, incluido su tipo y el estado del programa donde ocurrió.
- El sistema de ejecución recorre la pila de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



Puntos de vista ante las excepciones

- Hay dos puntos de vista para las excepciones:
 - El que lanza (eleva o propaga) la excepción
 - Se encuentra en una situación que no sabe cómo (o quiere) resolver y crea una excepción y la lanza
 - Le llega la excepción y no sabe qué hacer con ella y la propaga
 - El que la trata (captura) la excepción
 - Sabe cómo resolver la situación que ha provocado una excepción y dispone de un tratamiento para ella.
- Un método puede actuar desde los dos puntos de vista
 - Captura unas excepciones y lanza otras.
- Las especificaciones suelen indicar 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 JVM o con una instrucción **throw**, y sólo éstos pueden ser argumentos de una cláusula **catch**.
- Por convención, la clase **Throwable** y sus subclases tienen dos constructores:
 - Sin argumentos y
 - Con un argumento de tipo **String**, el cual puede ser usado para indicar 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 *printWriter* especificado.

Lanzar una excepción

- Una excepción es una instancia de una clase que se crea con new.
- Para lanzarla se utiliza
`throw objExcepción`
- Esto interrumpe el flujo de ejecución y se procede a la búsqueda de un manejador para esa excepción, es decir, alguien que la trate.

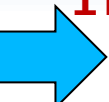
```
throw new RuntimeException("comentario adecuado");
```

```
public void Libro(String aut, String tit, double prec) {  
    if (prec < 0) {  
        throw new RuntimeException("El precio no puede ser negativo");  
    }  
    autor = aut;  
    titulo = tit;  
    precioBase = prec;  
}
```

Lanzar una excepción

En la clase `LibreriaOferta` del ejemplo `prLibreria`

```
public void setOferta(double desc, String[] autores) {  
    if (desc < 0)  
        throw new RuntimeException("Descuento negativo");  
    porcDescuento = desc;  
}
```

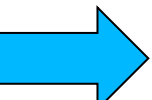


Hay que cambiar el descuento, ¡pero se pasa un número negativo!

¡No sabemos qué hacer! Lanzamos la excepción.

En la clase `Jarra` del proyecto `prJarras`

```
public class Jarra {  
    ...  
    public Jarra(int cap) {  
        if (cap <= 0)  
            throw new RuntimeException("Capacidad negativa");  
        capacidad = cap;  
    }  
}
```



¡Queremos crear una jarra con capacidad negativa!

¡No sabemos qué hacer! Lanzamos la excepción.

Propagación de excepciones

- Una excepción será automáticamente propagada si no se trata

```
public int convertirAEntero(String str) {  
    int n = Integer.parseInt(str);  
    return n;  
}
```

Si **str** no es convertible a entero se lanza una **NumberFormatException** y se propaga

Propagación de excepciones

- En la clase **PruebaJarras** del proyecto **prJarras**

```
public class PruebaJarras {  
    public static void main(String[] args) {  
        Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
        Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
        ...  
    }  
}
```

Si **args[0]** o **args[1]** no es convertible a entero: se lanza 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 PruebaJarras.main(TestUrna.java:5)

Captura de excepciones

La instrucción `try-catch`

```
try (  
    // Se crean objetos "closeables" que se  
    // automáticamente al terminar el bloque  
) {  
    //  
    // cuerpo vigilado  
    //  
} catch (Excepción11 | Excepción12 | ... | Excepción1N e) {  
    // Tratamiento común para todas las excepciones capturadas  
}  
catch (Excepción21 | Excepción22 | ...  
    // Tratamiento común para todas las excepciones capturadas  
...  
} catch (ExcepciónNN e) {  
    // Tratamiento común para todas las excepciones capturadas  
}
```

El bloque `try` vigila código para capturar las posibles excepciones que se produzcan.

Los bloques `catch` se ejecutan si se lanza una excepción adecuada en el bloque vigilado (son opcionales)

Es posible abrir objetos "closeable" que se cerrarán automáticamente cuando termine el bloque `try` (es opcional)

La captura de excepciones

```
public class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            ...  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras<Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            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 class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            ...  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras<Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            System.out.println("Los args deben ser enteros");  
        }  
    }  
}
```

1: Ocorre una excepción

ArrayIndexOutOfBoundsException

```
public class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            double proporcion = jarra1.getContenido() / jarra2.getContenido();  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras <Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            System.out.println("Los argumentos deben ser enteros");  
        } finally {  
            System.out.println("El programa sigue aquí");  
        }  
        System.out.println("Y después por aquí");  
    }  
}
```

Uso: PruebaJarras <Int> <Int>
El programa sigue aquí
Y después por aquí

2: Ocorre NumberFormatException

```
public class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            double proporcion = jarra1.getContenido() / jarra2.getContenido();  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras <Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            System.out.println("Los argumentos deben ser enteros");  
        } finally {  
            System.out.println("El programa sigue aquí");  
        }  
        System.out.println("Y después por aquí");  
    }  
}
```

Los argumentos deben ser enteros
El programa sigue aquí
Y después por aquí

3: El bloque try termina normalmente

```
public class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            double proporcion = jarra1.getContenido() / jarra2.getContenido();  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras <Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            System.out.println("Los argumentos deben ser enteros");  
        } finally {  
            System.out.println("El programa sigue aquí");  
        }  
        System.out.println("Y después por aquí");  
    }  
}
```

El programa sigue aquí
Y después por aquí

4: Ocorre

ArithmeticException

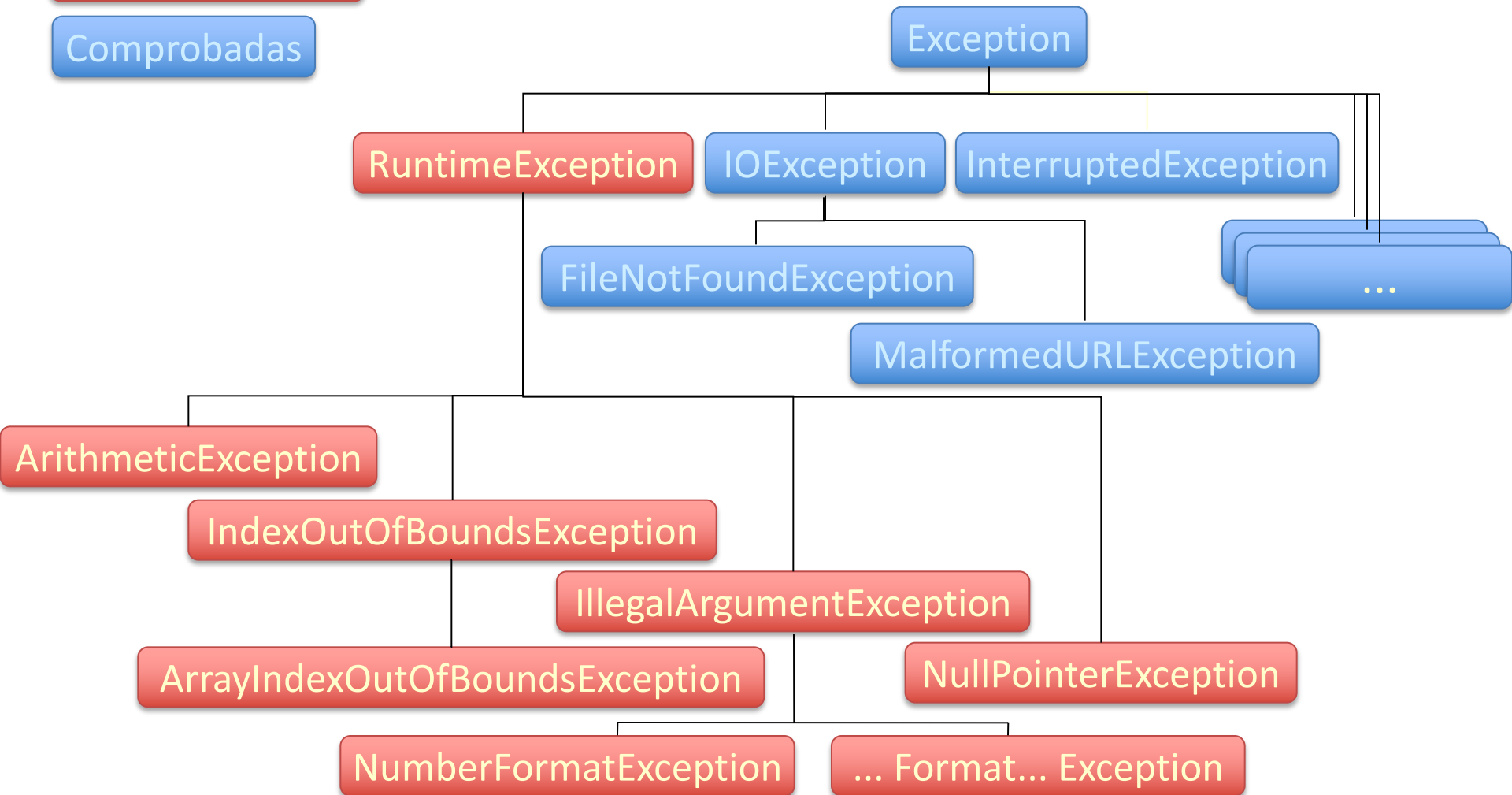
```
public class PruebaJarras {  
    public static void main(String[] args) {  
        try {  
            Jarra jarra1 = new Jarra(Integer.parseInt(args[0]));  
            Jarra jarra2 = new Jarra(Integer.parseInt(args[1]));  
            double proporcion = jarra1.getContenido() / jarra2.getContenido();  
        } catch (ArrayIndexOutOfBoundsException ae) {  
            System.out.println("Uso: PruebaJarras <Int> <Int>");  
        } catch (NumberFormatException nfe) {  
            System.out.println("Los argumentos deben ser enteros");  
        } finally {  
            System.out.println("El programa sigue aquí");  
        }  
        System.out.println("Y después por aquí");  
    }  
}
```

El programa sigue aquí

Las excepciones de obligado tratamiento (*checked* o comprobadas)

No comprobadas

Comprobadas



Excepciones comprobadas

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

```
public void leerFichero(String nombreFichero) {  
    try {  
        File fichero = new File(nombreFichero);  
        // Lectura del fichero  
    } catch (FileNotFoundException e) {  
        System.err.println("fichero no encontrado");  
    }  
}
```

Capturada

Propagación de excepciones comprobadas

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

```
public void leerFichero(String nombreFichero) throws FileNotFoundException {  
    File fichero = new File(nombreFichero);  
    // Leer fichero  
}
```

Anunciada

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

Excepciones relacionadas

```
public void escribeLista(int top) {  
    try {  
        PrintWriter out = new PrintWriter( "out.txt");  
        for (int i = 0; i < top; 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.

Nuevas funcionalidades en Java 1.7

Se modificó la sintaxis del bloque try

```
try (  
    // Se crean objetos closeable que se cerraran  
    // automáticamente al terminar el bloque try  
) {  
    // cuerpo vigilado  
} catch (Excepción1 | Excepción2 |  
        ... | ExcepciónN e) {  
    // Tratamiento común para todas las excepciones  
    // capturadas  
}
```

Reglas para tratar situaciones excepcionales

- **Preventiva:**

- La comprobación es poco costosa y es probable que se produzca la excepción.
- Ejemplo:
 - El constructor `Libro(String aut, String tit, double prec)`

- **Curativa:**

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

```
public int convertirAEntero(String str) {  
    int n = Integer.parseInt(str);  
    return n;  
}
```