Tratamiento de excepciones

Samsung
TECH INSTITUTE

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 { códig
    } else { códigoDe
    return códigoDeEr
}
```

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

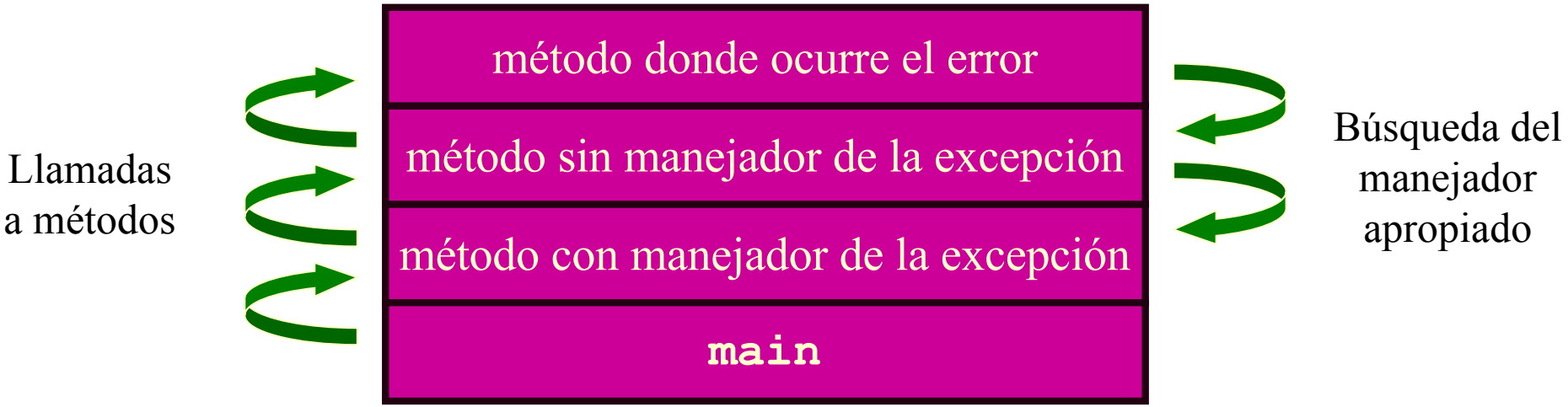
El tratamiento de excepciones

```
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
        ...;
    } 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.

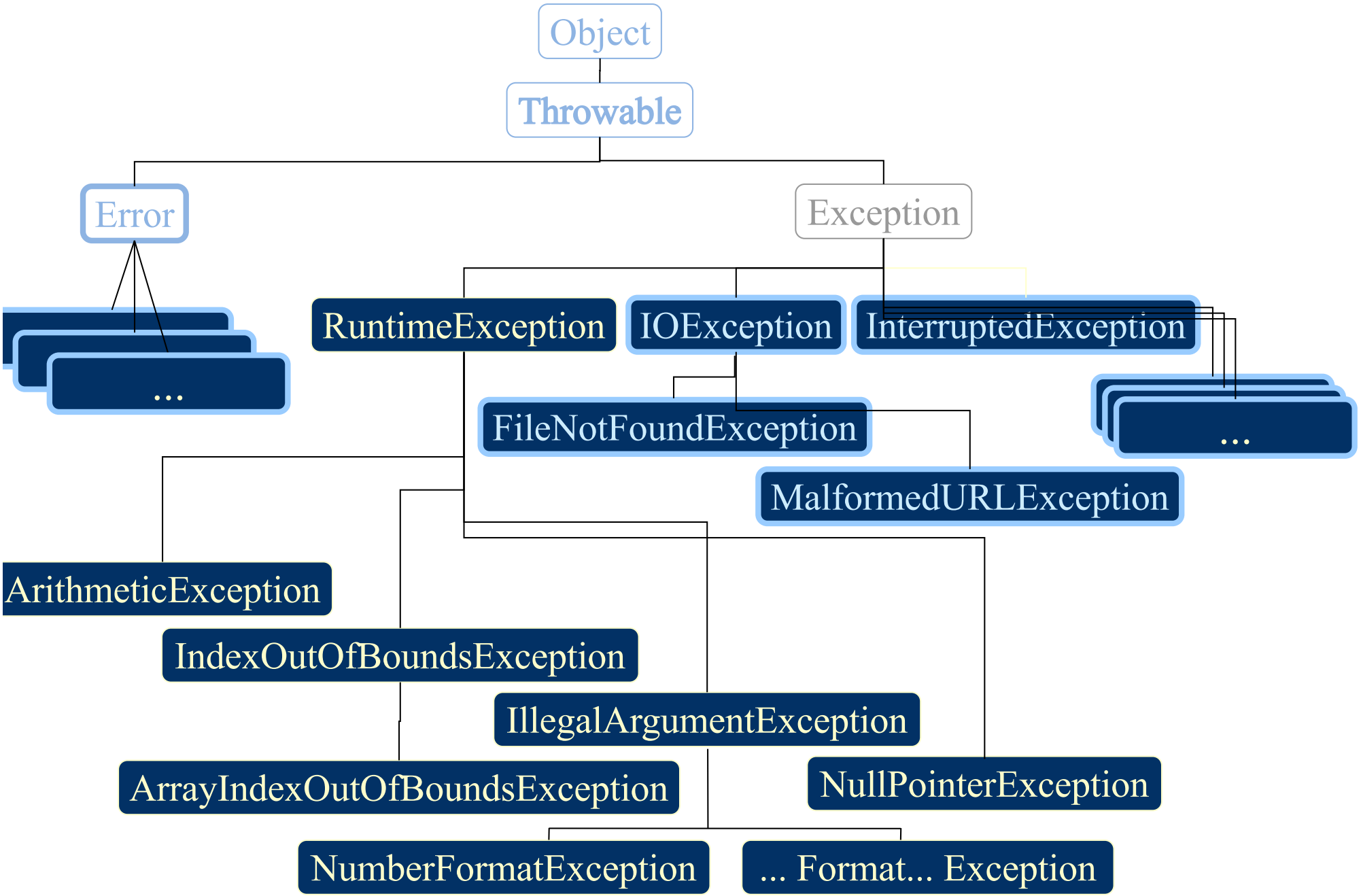
¿Qué es una excepción?

- 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 no 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 una situación que provoca 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.


Object
Throwable
Error
Exception
...
RuntimeException
IOException
InterruptedException
...
FileNotFoundException
MalformedURLException
ArithmeticException
IndexOutOfBoundsException
IllegalArgumentException
ArrayIndexOutOfBoundsException
NullPointerException
NumberFormatException
... Format... Exception

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 datos de este objeto y la traza de ejecución en la salida de errores estándar.

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 Jarra(int capacidadInicial) {
   if (capacidadInicial < 0 ) {
      throw new RuntimeException("Capacidad negativa");
   }
   capacidad = capacidadInicial;
}
```

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 el corte pero son paralelas!

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

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

¡Nos piden una bola y no hay!

Propagación de excepciones

- Una excepción será automáticamente propagada si no se trata

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

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

Propagación de excepciones

- 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 son convertibles 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 TestUrna.main(TestUrna.java:5)
```

Captura de excepciones
La instrucción `try-catch`

```
try (
    // Se crean objetos "closeables" que s
    // automáticamente al terminar el bloq
) {
    //
        // uerpo vigilado
    //
} catch ( epción11 | Excepción12 | ... | Excepción1N e)  {

    // Tra nto común para todas las ex

} catch ( ión21 | Excepción22 | ...

    // Tra  común para todas las

...

} ca .. | ExcepciónNN e)  {

    /  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 closeables que se cerrarán automáticamente cuando termine el bloque **`try`** (es opcional)

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 argumentos deben ser números enteros");
       }
    }
}
```

- Los manejadores pueden preguntar al usuario por la decisión a tomar, recuperarse automáticamente del error, terminar el programa, etc.
- 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 TestUrna {
   public static void main(String[] args) {
     try {
      int bb = Integer.parseInt(args[0]);
      int bn = Integer.parseInt(args[1]);
      int d = bb/bn;
     } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Uso: TestUrna <Int> <Int>");
     } catch (NumberFormatException ee) {
      System.out.println("Los argumentos deben ser enteros");
     } finally {
       System.out.println(”El programa sigue aquí”);
     }
     System.out.println(”Y después por aquí”);
   }
}
```

1: Ocurre una excepción
ArrayIndexOutOfBoundsException

```
public class TestUrna {
   public static void main(String[] args) {
     try {
      int bb = Integer.parseInt(args[0]);
      int bn = Integer.parseInt(args[1]);
      int d = bb/bn;
     } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Uso: TestUrna <Int> <Int>");
     } catch (NumberFormatException ee) {
      System.out.println("Los argumentos deben ser enteros");
     } finally {
       System.out.println(”El programa sigue aquí”);
     }
     System.out.println(”Y después por aquí”);
   }
}
```

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

2: Ocurre NumberFormatException

```
public class TestUrna {
   public static void main(String[] args) {
     try {
      int bb = Integer.parseInt(args[0]);
      int bn = Integer.parseInt(args[1]);
      int d = bb/bn;
     } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Uso: TestUrna <Int> <Int>");
     } catch (NumberFormatException ee) {
      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 TestUrna {
   public static void main(String[] args) {
     try {
      int bb = Integer.parseInt(args[0]);
      int bn = Integer.parseInt(args[1]);
      int d = bb/bn;
     } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Uso: TestUrna <Int> <Int>");
     } catch (NumberFormatException ee) {
      System.out.println("Los argumentos deben ser enteros");
     } finally {
       System.out.println(”El programa sigue aquí”);
     }
   }
   System.out.println(”Y después por aquí”);
}
```

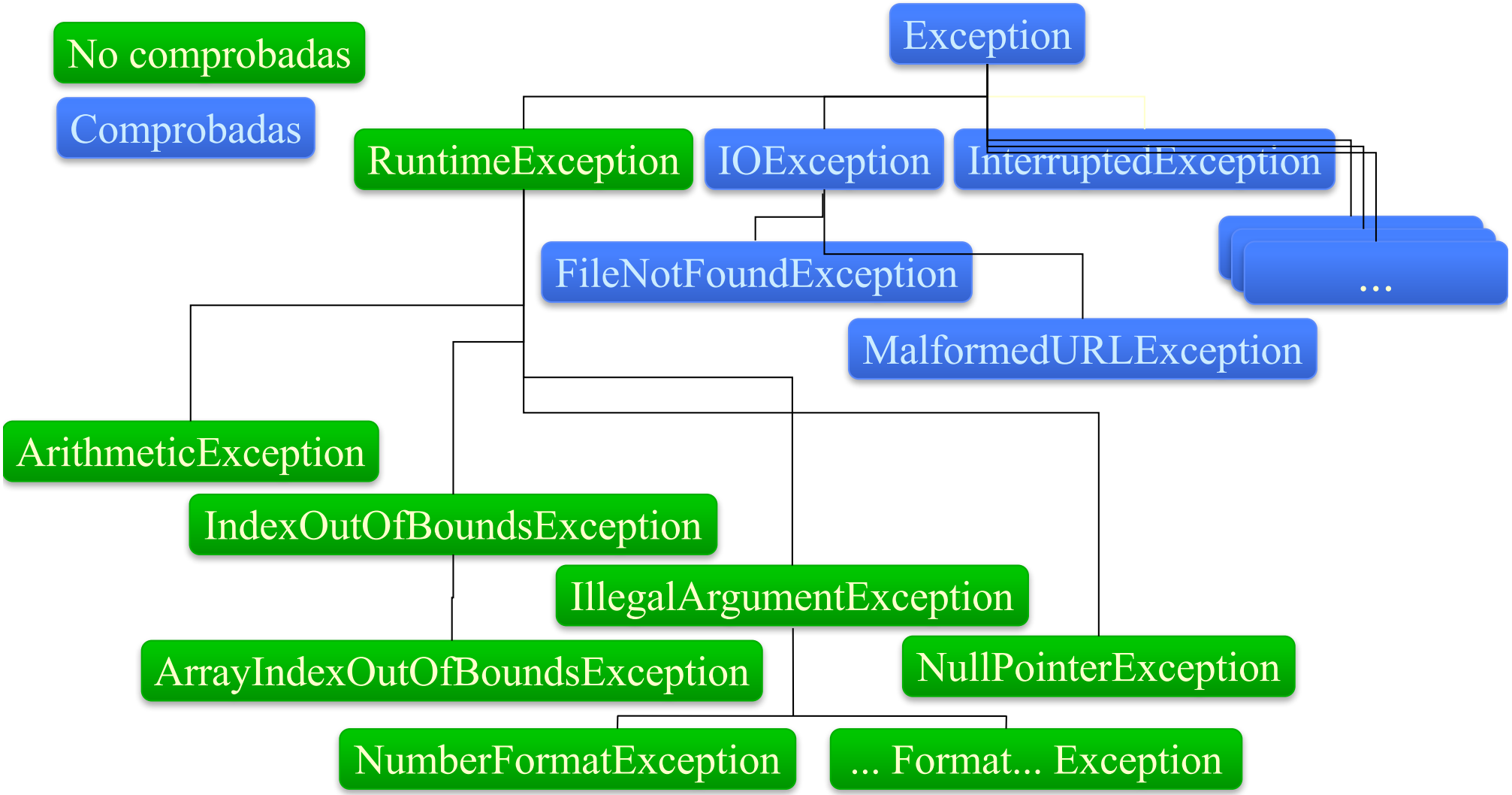
```
El programa sigue aquí
Y después por aquí
```

4: Ocurre ArithmeticException

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



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 espera(int millis) {
    try {
            Thread.sleep(millis)
    } catch (InterruptedException e) {
            System.err.println(”se interrumpió");
    }
}
```

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 espera(int millis) throws InterruptedException {
        Thread.sleep(millis)
}
```

Anunciada

- 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 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 argumentos deben ser enteros");
     } catch (RuntimeException e) {
      System.out.println(”Alguna excepción distinta");
     } finally {
       System.out.println(”El programa sigue aquí”);
     }
   }
}
```

Definiendo nuestras propias excepciones

Un usuario puede definir sus propias excepciones.

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

Y ahora puede lanzarse como las demás.

```
throw new MiExcepcion(5);
```

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) {
  return Integer.parseInt(str);
}
```