# Capítulo 14
# Manejo de Excepciones

# Motivaciones

Cuando un programa se encuentra con un error en tiempo de ejecución, el programa termina anormalmente.

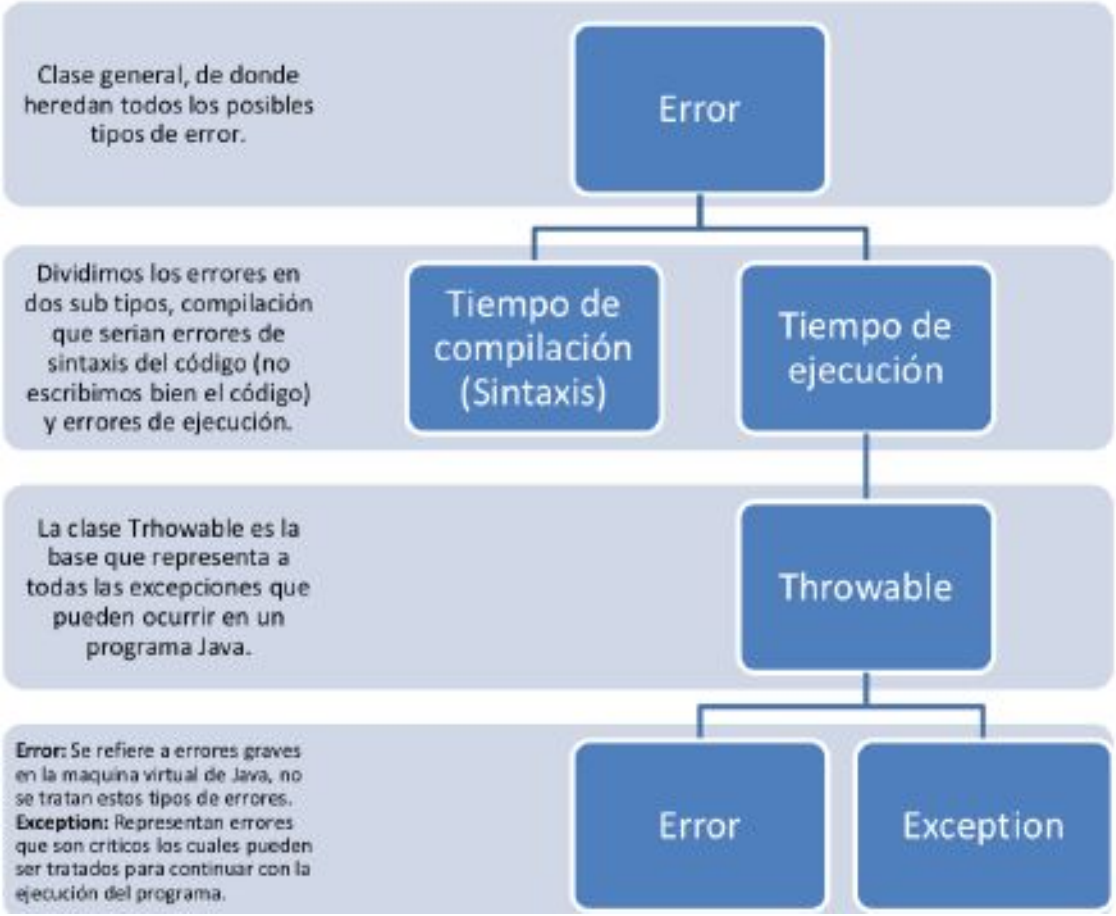
¿Cómo podemos manejar el error en tiempo de ejecución para que el programa pueda seguir ejecutándose o finalizar correctamente?

Este es el tema que presentaremos en este capítulo

# Objetivos

- Obtener una descripción general de las excepciones y el manejo de excepciones.
- Explorar las ventajas de usar el manejo de excepciones.
- Distinguir los tipos de excepción: Error (fatal) vs Excepción (no fatal) y verificado vs. No verificado.
- Declarar excepciones en un encabezado de método.
- Lanzar excepciones en un método.
- Escribir un bloque try-catch para manejar excepciones.
- Explicar cómo se propaga una excepción.
- Obtener información de un objeto de excepción.
- Desarrollar aplicaciones con manejo de excepciones.
- Usar la cláusula finally en un bloque try-catch.
- Usar excepciones sólo para errores inesperados.
- Volver a lanzar excepciones en un bloque catch.
- Crear excepciones encadenadas.
- Definir clases de excepción personalizadas.

# Tipos de errores



# Tipos de errores

***Errores en tiempo de compilación***

Se suelen producir cuando el código no está bien escrito, por ejemplo cuando falta el punto y coma al final de la sentencia.

Hasta que no se arreglen, no se genera el archivo con la extensión .class

***Errores en tiempo de ejecución. Excepciones***

El programa compila y se puede ejecutar pero, por algún motivo, se produce un fallo (errores físicos, fallos de dispositivos, operaciones no permitidas, datos incorrectos, errores de seguridad) y el programa se “rompe”. Es necesario preveerlos para que el programa no termine de forma inesperada.

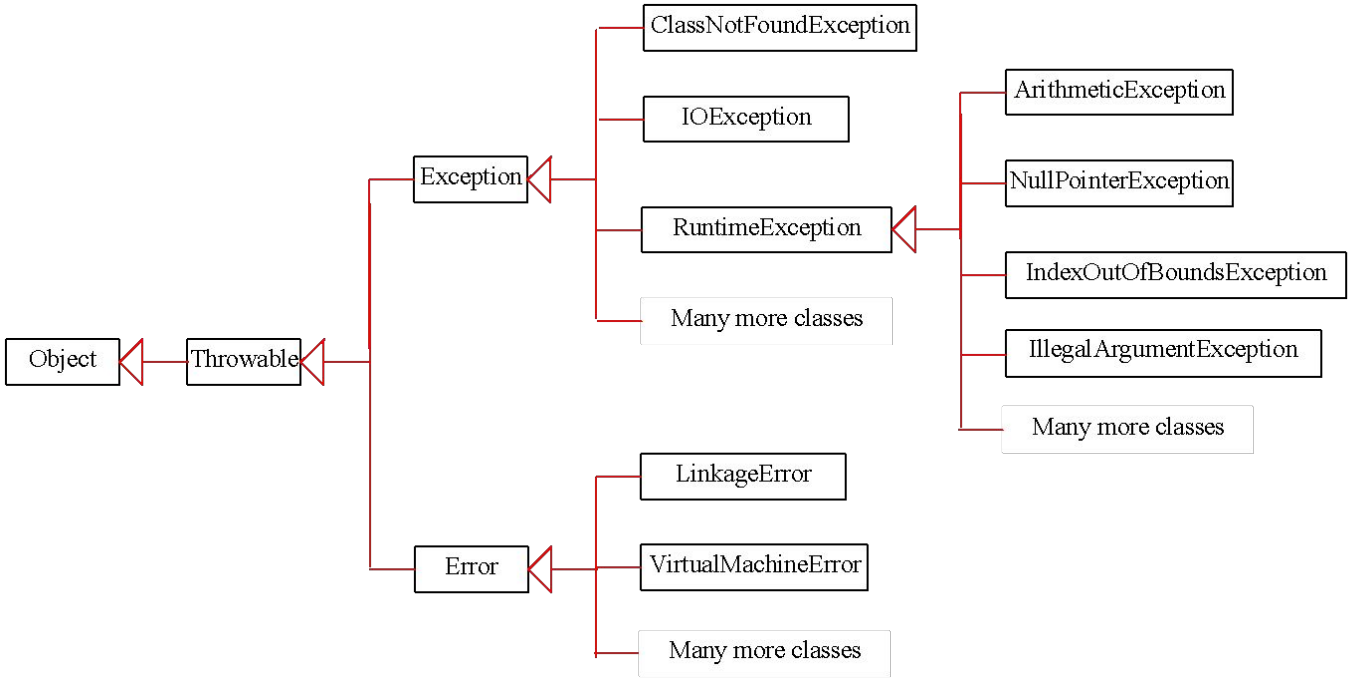
# Excepciones

Los distintos tipos de errores se representan mediante clases de excepción, que son clases Java que heredan directa o indirectamente de la clase Throwable.

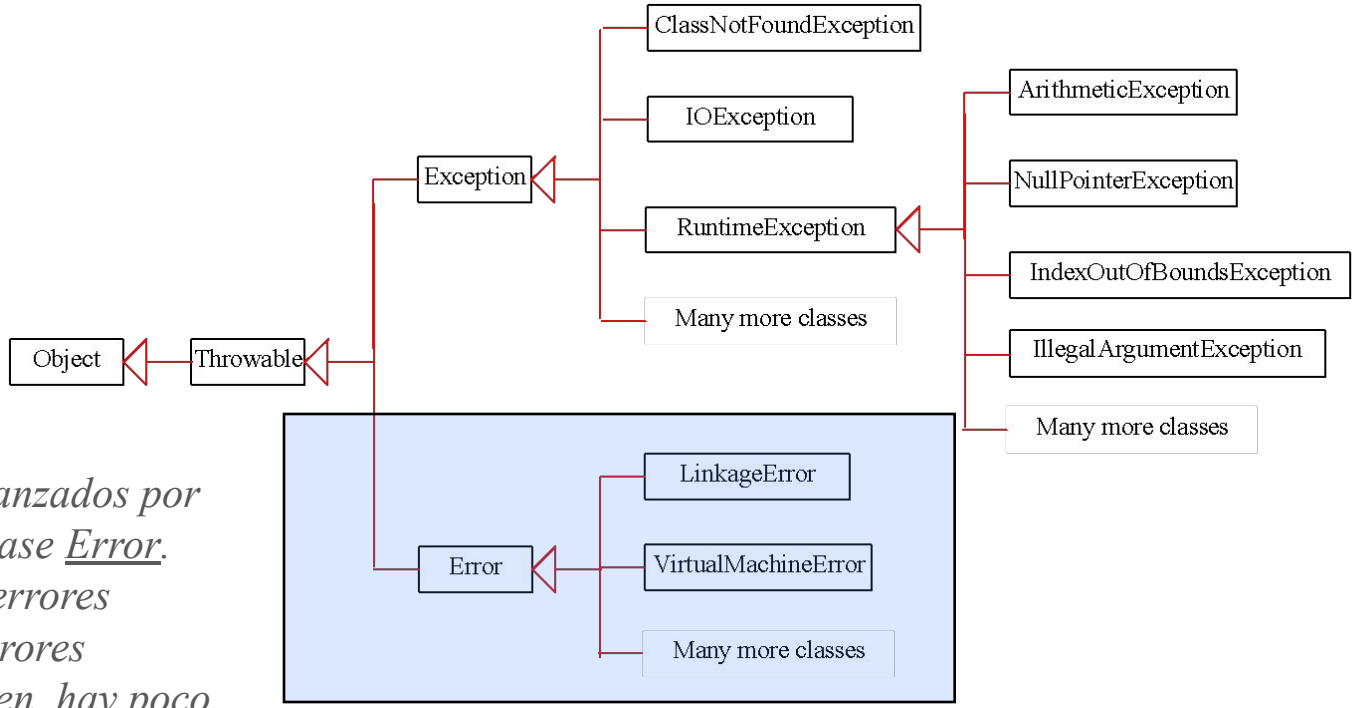
De la clase Throwable heredan dos clases: Error y Exception. La primera representa errores o situaciones “intratables” desde el programa como fallos de la máquina virtual y la segunda representa los errores que en mayor o menor medida pueden ser tratados desde el programa.

Existen tipos de excepciones que son conocidas por el compilador (porque los métodos declaran que pueden lanzarlas), llamadas checked exceptions, y que estamos obligados a tratar o propagar, y otras que podemos tratar sólo si prevemos que se van a producir.

# Tipos de Excepciones



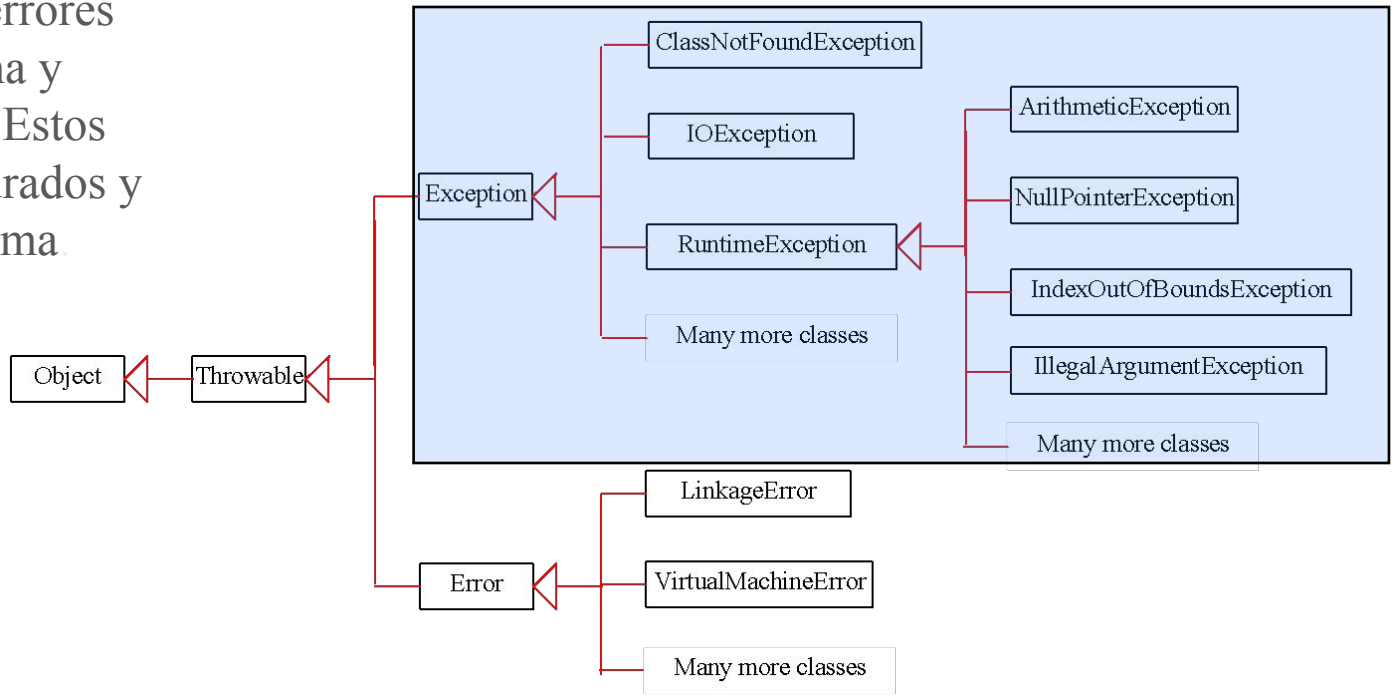
# Errores de Sistema



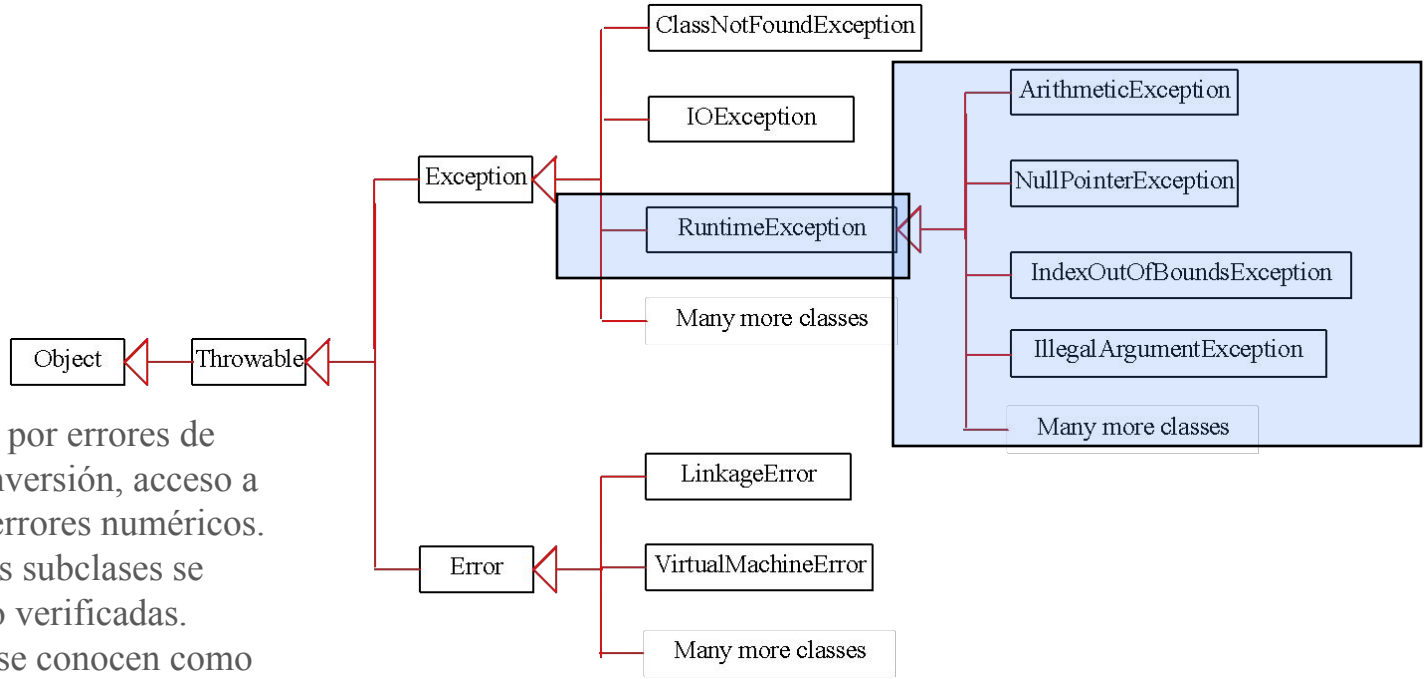

*Los errores del sistema son lanzados por JVM y representados en la clase Error. La clase Error describe los errores internos del sistema. Tales errores raramente ocurren. Si lo hacen, hay poco que se pueda hacer más allá de notificarlo al usuario y tratar de terminar el programa correctamente.*

# Exceptions

Exception describe los errores causados por el programa y circunstancias externas. Estos errores pueden ser capturados y manejados por el programa.



# Excepciones Runtime




RuntimeException es causada por errores de programación, como mala conversión, acceso a una matriz fuera de límites y errores numéricos. RuntimeException, Error y sus subclases se conocen como excepciones no verificadas. Todas las demás excepciones se conocen como excepciones verificadas, lo que significa que el compilador obliga al programador a verificar y tratar las excepciones.

# Excepciones No Verificadas

En la mayoría de los casos, las excepciones no verificadas reflejan errores de lógica de programación que no son recuperables.
Pueden ocurrir en cualquier parte del programa.
Para evitar el uso excesivo de los bloques try-catch, Java no le obliga a escribir código para capturar las excepciones no verificadas.

Ejemplo:

- se lanza una NullPointerException si accede a un objeto a través de una variable de referencia antes de que se le asigne un objeto;

- se lanza una excepción IndexOutOfBoundsException si accede a un elemento en una matriz fuera de los límites de la matriz.

# Declarar, lanzar y atrapar excepciones

Para controlar las posibles excepciones en un bloque de código Java se usa la estructura try-catch-finally, que tiene la siguiente sintaxis:

```
try {
        acciones;
} catch (TipoExcepcion var) {
        tratar_excepcion;
}
…
finally {
        acciones_incondicionales;
{
```

Una vez ejecutado el bloque catch, la excepción expira, y la ejecución del programa continuará de forma normal a partir de la siguiente instrucción a la estructura try-catch.

# Declarar, lanzar y atrapar excepciones

El **bloque try (o** bloque de acciones) es el que contiene las instrucciones que queremos ejecutar, y que potencialmente pueden recibir una excepción.
Si se produce una excepción durante la ejecución de dichas sentencias, se interrumpe la ejecución del bloque y el programa salta al bloque catch correspondiente a la excepción recibida. Este bloque de control de excepciones puede contener tantas sentencias como queramos.

El **bloque catch** (o bloque de tratamiento de excepciones) es el encargado de tratar la excepción del tipo especificado como argumento, si se produce. Sus sentencias irán encaminadas a solucionar el problema o dejar constancia de él, de forma que el programa pueda continuar con normalidad.

# Declarar, lanzar y atrapar excepciones

Podemos incluir tantos **bloques catch** como queramos para un try, siempre que traten distintos tipos de excepción. Podemos elegir el tipo de excepción que mejor se adecúe al error que queremos representar de entre todas las clases de excepción de la API.

```
try {
 declaraciones; //Declaraciones que pueden arrojar excepciones
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

# Declarar, lanzar y atrapar excepciones

El **bloque finally** es opcional, y sirve para realizar una salida segura, liberando recursos, etc.
Este bloque se ejecutará al final de la sentencia tanto si se ha ejecutado todo el bloque try como si ha ocurrido una excepción y se ha ejecutado el bloque catch.

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

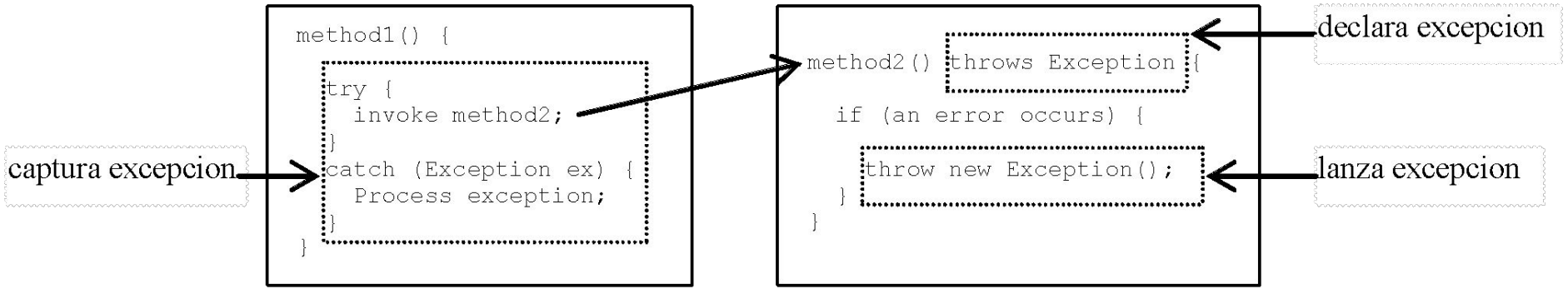
# Declarar, lanzar y atrapar excepciones

```
public class EjemploExcepciones {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Este programa calcula la media de dos números");
        try {
                System.out.print("Introduzca el primer número: ");
                double numero1 = sc.nextDouble();
                System.out.print("Introduzca el segundo número: ");
                double numero2 = sc.nextDouble();
                System.out.println("La media es " + (numero1 + numero2) / 2);
        } catch (Exception e) {
                System.out.println("Excepción: " + e.getClass());  //Except: java.lang.NumberFormatException
                System.out.println("Error: " + e.getMessage());    //Error: For input string: "hola"
                System.out.print("No se puede calcular la media. ");
                System.out.println("Los datos introducidos no son correctos.");
        }
    }
}
```

# Propagar excepciones

Otra opción es propagar las excepciones en lugar de tratarlas. Para ello tendremos que declarar en el método, que lanza la excepción o excepciones adecuadas.

```
void metodo(lista_de_parámetros) throws TipoExc1, TipoExc2 {
acciones;
}
```



# Declarar y lanzar Excepciones

Cada método debe indicar los tipos de excepciones verificadas que podría lanzar. Esto se conoce como ***declarar excepciones***.

```
public void myMethod()
  throws IOException
public void myMethod()
  throws IOException, OtherException
```

Cuando el programa detecta un error, el programa puede crear una instancia de un tipo de excepción apropiado y lanzarlo. Esto se conoce como ***lanzar una excepción***. Este es un ejemplo,

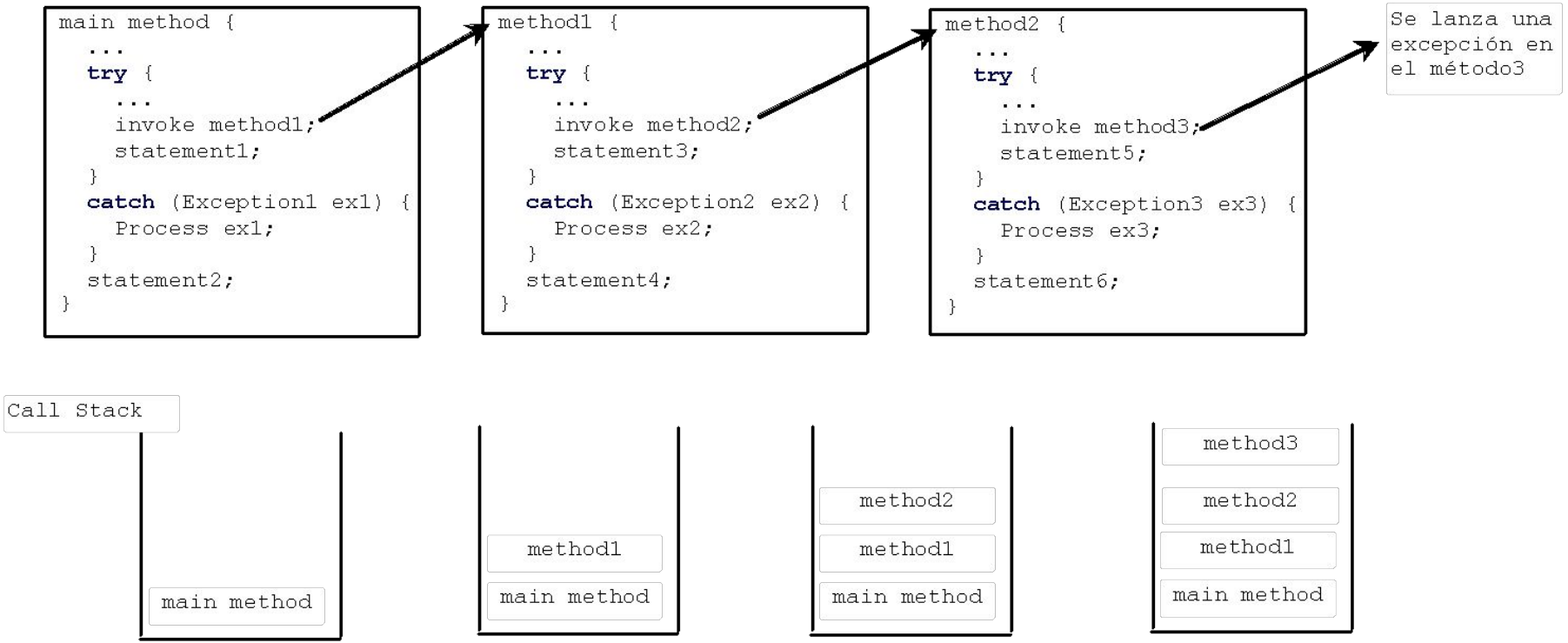
```
throw new TheException();
TheException ex = new TheException();
throw ex;
```

# Ejemplo lanzamiento de Excepciones

Con la sentencia **throw** podemos lanzar una excepción para interrumpir la ejecución de nuestro método y hacer que el método que llamó reciba la excepción para poder tratarla adecuadamente. Para ello crearemos un objeto de la clase de excepción deseada y si lo deseamos, le añadiremos un mensaje de error, y con la sentencia throw, lanzaremos la excepción.

```
  /** Set a new radius */
public void setRadius(double newRadius)
    throws IllegalArgumentException {
  if (newRadius >= 0)
    radius =  newRadius;
  else
    throw new IllegalArgumentException(
      "Radius cannot be negative");
}
```

# Captura de Excepciones



# Lanzamiento de excepciones con throw

La orden **throw** permite lanzar de forma explícita una excepción.

Por ejemplo, la sentencia *throw new ArithmeticException()* crea de forma artificial una excepción:

```
public class EjemploExcepcionesthrow {
    public static void main(String[] args) {
        System.out.println("Inicio");
        throw new ArithmeticException();
    }
}
```

Throw también es útil cuando se recoge la excepción en un método y luego, esa misma excepción se vuelve a lanzar para que la recoja, a su vez, otro método y luego otro y así sucesivamente hasta llegar al main.

# Ejemplo excepciones en main y función

```
public class EjemploExcepManzanasConThrow {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Número de manzanas: ");
        int m = sc.nextInt();
        System.out.print("Número de personas: ");
        int p = sc.nextInt();
        try {
            System.out.print("A cada persona le corresponden " + reparteManzanas(m, p) + " manzanas.");
        } catch (ArithmeticException ae) {
            System.out.println("Los datos introducidos no son correctos.");
        }
    }
    public static int reparteManzanas(int manzanas, int personas) {
        try {
            return manzanas / personas;
        } catch (ArithmeticException ae) {
            System.out.println("El número de personas vale 0.");
            throw ae;
        }
    }
}
```

# Capturar o declarar excepciones validadas

Supongamos que p2 se define de la siguiente manera :

```
void p2() throws IOException {
  if (a file does not exist) {
     throw new IOException("File does not exist");
  }

  ...
}
```

Java obliga a lidiar con excepciones validadas. Si un método declara una excepción validada (es decir, una excepción que no sea de tipo Error o RuntimeException), debemos invocarla en un bloque try-catch o declarar- lanzar la excepción en el método de llamada.

# Capturar o declarar excepciones validadas

Por ejemplo, supongamos que el método p1 invoca el método p2 y p2 puede arrojar una excepción validada (Ej, IOException), debe escribir el código como se muestra en (a) o (b).

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

# Traza de ejecución del programa

```
try {
  statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

Supongamos que no hay excepciones en las declaraciones

# Traza de ejecución del programa

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

El bloque final siempre se ejecuta

# Traza de ejecución del programa

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

Se ejecuta la siguiente instrucción en el método

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Supongamos que se arroja una excepción de tipo Exception1 en statement2

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

La excepción es manejada.

# Traza de ejecución del programa

```
try {
   statement1;
   statement2;
   statement3;
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

El bloque final siempre se ejecuta tanto si se producen excepciones como si no

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

La siguiente instrucción en el método se ejecuta ahora.

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}
Next statement;
```

statement2 arroja una excepción de tipo Exception2.

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}
Next statement;
```

Manejo de excepción

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}
Next statement;
```

Ejecuta el bloque final

# Traza de ejecución del programa

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}
Next statement;
```

Volver a lanzar la excepción y el control se transfiere a quien hizo la llamada

# Precauciones al usar excepciones

El manejo de excepciones separa el código de manejo de errores de las tareas normales de programación, lo que hace que los programas sean más fáciles de leer y modificar. Tengamos en cuenta, sin embargo, que el manejo de excepciones generalmente requiere más tiempo y recursos, ya que requiere instanciar un nuevo objeto de excepción, deshacer la pila de llamadas y propagar los errores a los métodos de llamadas.

# Cuando lanzar usar Excepciones

Si una excepción ocurre en un método y queremos que la excepción sea procesada por el llamador del método, tendremos que crear una instancia de esa excepción y lanzarla. Si podemos manejar la excepción en el método donde ocurre, no hay necesidad de lanzarla.

¿Cuándo deberíamos usar el bloque try-catch en el código? Debe usarlo para tratar condiciones de error inesperadas. No se debe usar para tratar situaciones simples y esperadas.

# Ejemplo de Cuándo usar excepciones

Por ejemplo, el siguiente código

```
try {
  System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
  System.out.println("refVar is null");
}
```

es mejor reemplazarlo por

```
if (refVar != null)
  System.out.println(refVar.toString());
else
  System.out.println("refVar is null");
```

# Clases de excepción personalizadas

Use las clases de excepción en la API siempre que sea posible.

Defina clases de excepción personalizadas si las clases predefinidas no son suficientes y necesita tener tipos de errores específicos para su aplicación.

Defina clases de excepciones personalizadas extendiendo Exception o una subclase de Exception. Sólo hay que definir una clase que herede de la clase de excepción donde mejor encaje nuestro error como subtipo. Sería interesante añadirle al menos dos constructores: uno por defecto y otro que tome como parámetro una cadena con el mensaje y se lo pase al constructor de la superclase.

# Ejemplo excepciones personalizadas

Para crear una nueva Exception personalizada debemos seguir los siguientes pasos:

1. Crear una nueva clase
2. Extender Exception u otra clase descendiente de esta.
3. Creación de los constructores necesarios.

```
public class ExcepcionAlturaFueraDeRango extends Exception {
	public ExcepcionAlturaFueraDeRango(){
		System.out.println("ExcepcionAlturaFueraDeRango: La altura está fuera del rango permitido.");
	}
}
```

Y ya podemos utilizar la clase de excepción personalizada

# Ejemplo excepciones personalizadas

```
public class PruebaExcepcionesPropias {
  public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca la altura de la pirámide (un número entre 1 y 10): ");
        int h = sc.nextInt();
        try {
              pintaPiramide(h);
        } catch (ExcepcionAlturaFueraDeRango eafr) {
              System.out.println("No se ha podido pintar la pirámide.");
        }
  }
  public static void pintaPiramide(int h) throws ExcepcionAlturaFueraDeRango {
        if ((h < 1) || (h > 10)) {
              throw new ExcepcionAlturaFueraDeRango();
        }
        …
  }
}
```