

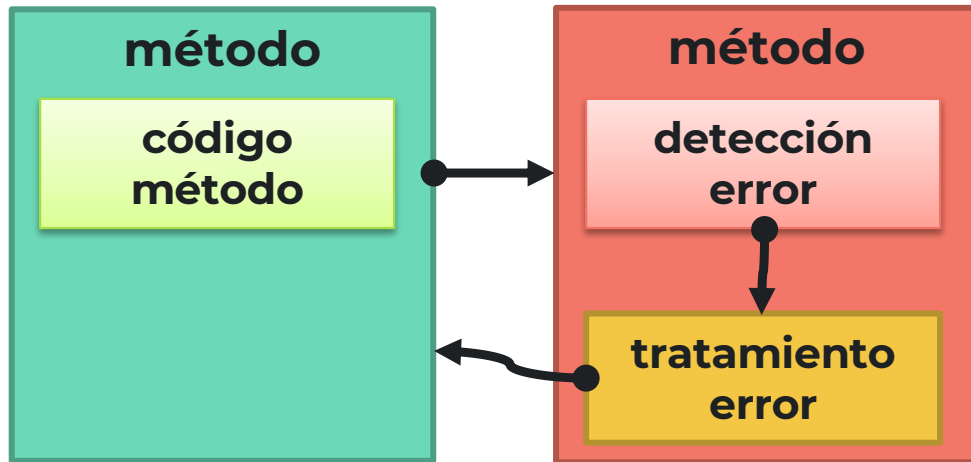
# 7. Excepciones

Gestión de errores y excepciones

# Concepto de excepción

- Una de las tareas importantes que es necesario realizar en el desarrollo de un programa es la **gestión de los errores** que ocurren durante su ejecución, ya que su correcto tratamiento facilita la usabilidad y mantenimiento del programa
- En la programación de los métodos se debe de favorecer la **separación lógica** de los diferentes tipos de código
  - Código asociado a la funcionalidad provista por el método
  - Código asociado a la interacción con usuarios u otros programas
  - Código asociado al almacenamiento y acceso a los datos
  - **Código asociado a la gestión de errores**

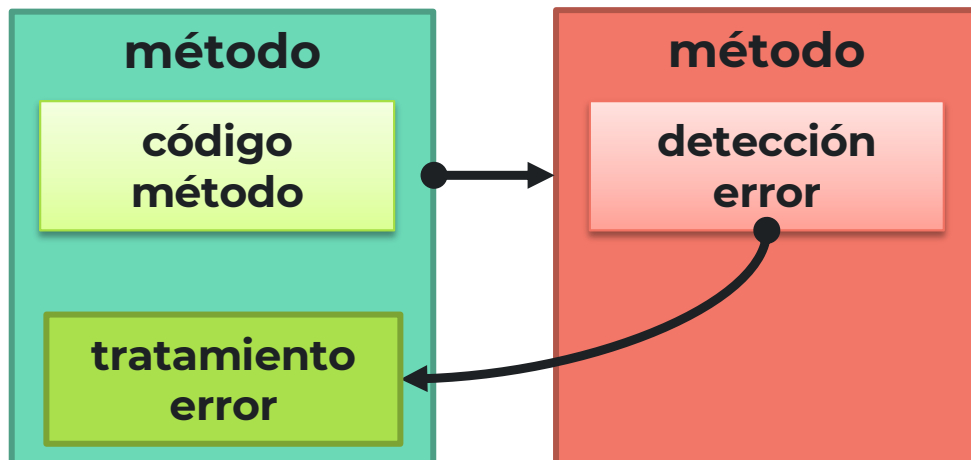
# Concepto de excepción



La detección y el tratamiento del error **están mezclados** con el código del método rojo, haciéndolo mucho más complejo entender y mantener

El método verde únicamente se encarga de invocar al método rojo

1

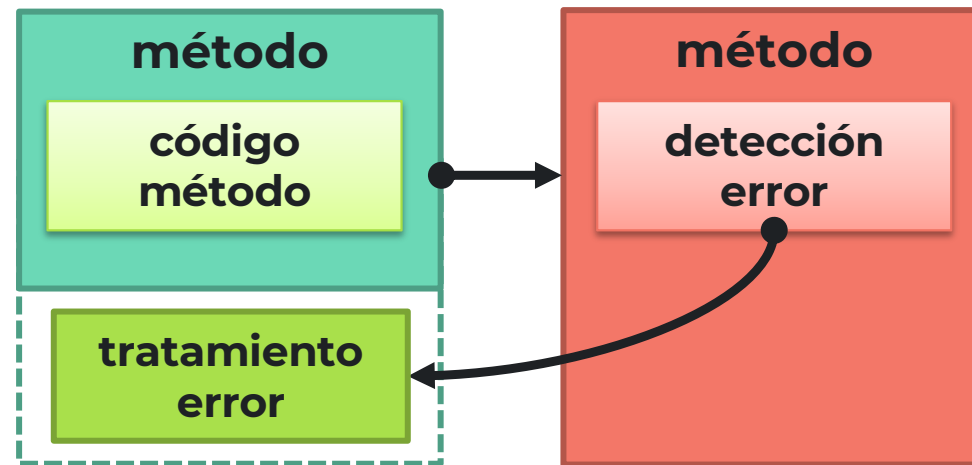


La detección del error forma parte del código del método rojo, pero es una **parte muy pequeña** del código, lo que simplifica el método si lo comparamos con la versión (1)

El método verde **mezcla** el código de tratamiento del error con su código

2

# Concepto de excepción



En el código método rojo se mantiene la detección del error para sacar ventaja de la simplificación en su código en relación a la versión (1)

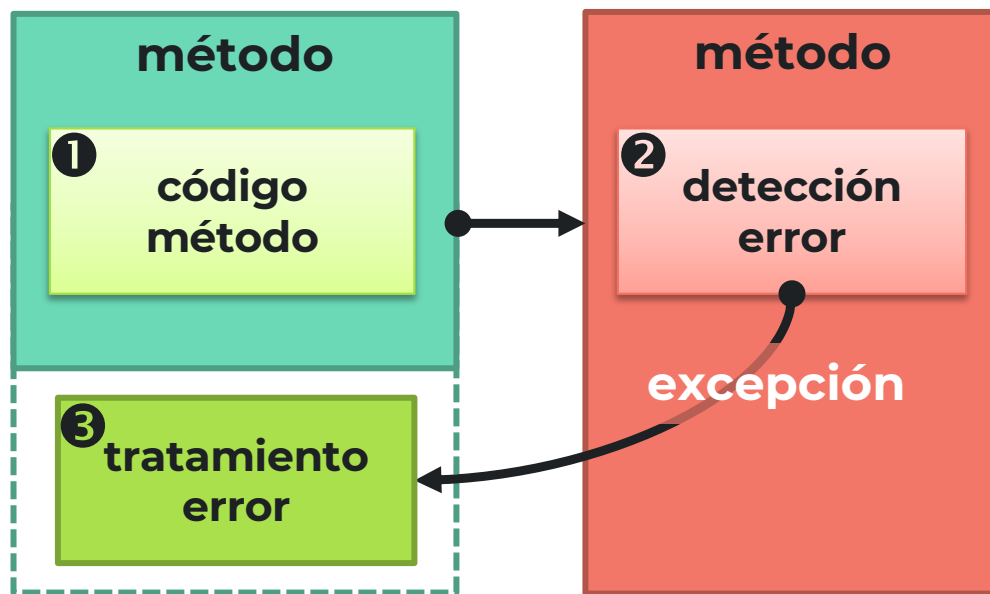
El tratamiento del error continua en el código del método verde, pero **existe una clara separación** entre el código de tratamiento del error y el código que proporciona la funcionalidad del método verde. De este modo, un **cambio** en uno de los dos tipos de código **no afectaría al otro**, facilitando enormemente su mantenimiento y legibilidad por parte de los programadores

# Concepto de excepción

- Una **excepción** es la ocurrencia de errores y/o situaciones de interés **durante la ejecución** de los métodos que proporcionan la funcionalidad del programa
  - Las excepciones no solo están relacionadas con lo que se entiende habitualmente por errores, sino también con **situaciones** a las que se debe dar respuesta y **que forman parte de la lógica del programa**
    - Una división en la que el cociente es cero **es un error**
    - La introducción de un comando incompleto **no es un error**, sino una condición que puede tener lugar en la ejecución del programa y que es necesario tratar

# Concepto de excepción

- El mecanismo de Java para el tratamiento de las excepciones aplica la filosofía de **separar el código** para la detección, la comunicación y el tratamiento de las excepciones **del código** de los métodos que proporciona la funcionalidad del programa



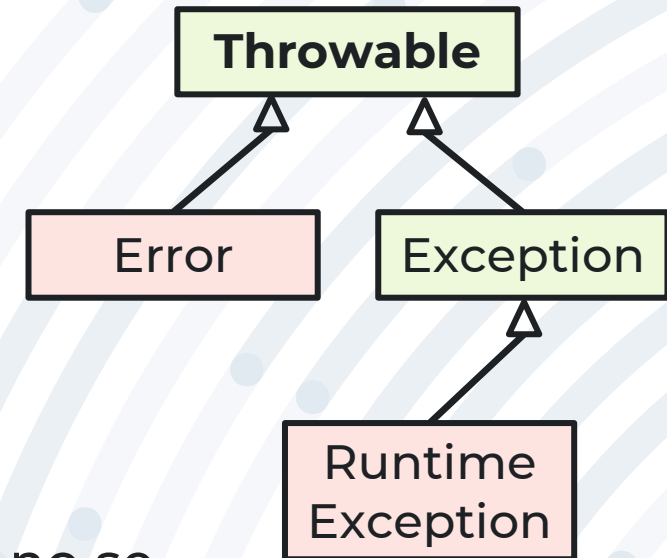
- 1 El método verde **intenta** ejecutar el método verde
- 2 La ejecución del método rojo **detecta y genera** una excepción, que comunica al método verde
- 3 El método verde **captura y trata** el error en un parte separada del código invocación

# Definición de excepciones

- Antes de especificar las etapas correspondientes a la gestión de las excepciones en tiempo de ejecución es necesario **(1)** definir qué excepciones pueden tener lugar; y **(2)** definir en qué métodos ocurrirán dichas excepciones
  - Una excepción se especifica como una clase derivada de la clase **Throwable**, cuyos métodos más usados habitualmente son los siguientes
    - **String getMessage( )** devuelve el mensaje que se ha indicado en la ocurrencia de la excepción
    - **void printStackTrace( )** imprime en consola la secuencia de invocaciones de métodos que ha llevado a la ocurrencia de la excepción

# Definición de excepciones

- En Java **existen dos tipos de excepciones**
  - **Excepciones chequeadas** (“checked”), se comprueba en tiempo de compilación en qué métodos puede ocurrir una excepción, en cuyo caso **esos métodos deben indicar de forma explícita** esa posible ocurrencia. **Son clases derivadas de la clase Exception**
  - **Excepciones no chequeadas** (“unchecked”), no se comprueba en tiempo de compilación cuáles son los métodos en los que tiene lugar la ocurrencia de la excepción, descargando en **el programador la responsabilidad** de chequear dónde puede ocurrir para intentar capturarla y tratarla. **Son clases derivadas de las clases RuntimeException o Error**





# Definición de excepciones

Excepciones de Java chequeadas	Excepciones de Java no chequeadas
<b>IOException</b> <b>SQLException</b> <b>URISyntaxException</b> ...	<b>ArithmeticException</b> <b>ClassCastException</b> <b>IndexOutOfBoundsException</b> <b>NullPointerException</b> <b>UnsupportedOperationException</b> ...

- Si el error o la condición que ha provocado la ocurrencia de la excepción **no impide** la ejecución del programa, la excepción debería ser del tipo chequeada
  - **Ejemplo:** NullPointerException es no chequeada porque en pocas ocasiones se puede **"arreglar"** un nulo de un objeto

# Definición de excepciones

- La definición de excepciones como **clases derivadas de la clase Exception** confiere a los programadores un mayor control sobre la gestión de dichas excepciones, puesto que obliga a que los métodos declaren qué excepciones podrían ocurrir durante su invocación
  - *public Exception(**String message**)*

**message** es el mensaje con el que típicamente el programador explica por qué ha tenido lugar la ocurrencia de la excepción, que es el momento en el que se crea el objeto de tipo clase derivada de Exception. Este mensaje es lo que habitualmente se le muestra al usuario
  - Este constructor **debería ser invocado** desde el constructor de la clase derivada de Exception

# Definición de excepciones

```
public class SueldoMinimo extends Exception {  
    // Atributos (de contexto)  
    private Empleado empleado;  
  
    // Constructor de la excepción  
    public SueldoMinimo(String mensaje, Empleado empleado) {  
        super(mensaje);  
        this.empleado = empleado;  
    }  
  
    public Empleado getEmpleado() {  
        return empleado;  
    }  
  
    public void setEmpleado(Empleado empleado) {  
        this.empleado = empleado;  
    }  
}
```

**SueldoMinimo** es una excepción que se lanzará cuando se intente escribir un valor para el sueldo del empleado que sea **menor** que el sueldo mínimo interprofesional

El atributo **empleado** se utiliza para pasar información de contexto entre el código que intenta la ejecución del método y el código que lleva a cabo su tratamiento

Desde el constructor de la excepción se **debe llamar** al constructor de la clase **Exception** para que el mensaje que se pase como argumento se obtenga con el método **getMessage** heredado de Throwable

# Definición de excepciones

- Para las excepciones chequeadas, el compilador exige que se indiquen **explícitamente** los métodos en los que tienen lugar dichas excepciones
  - Se deben de **etiquetar** los métodos en los que tienen lugar las excepciones
  - Un mismo método puede estar etiquetado **con más de un tipo de excepción**, es decir, con más de una clase derivada de Exception
  - Un **constructor también puede generar** excepciones

```
<tipo_acceso> <tipo_dato> nombre_método(<args> *) throws <excepción>  
<tipo_acceso> nombre_clase(<args> *) throws <excepción>
```

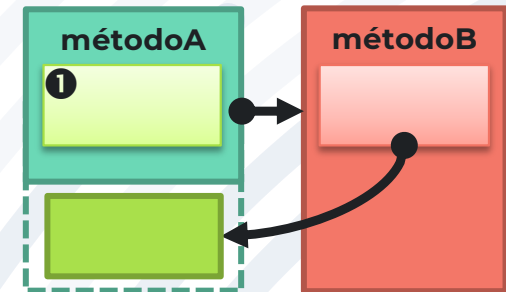
# Definición de excepciones

```
public class setSueldo(float sueldo) throws SueldoMinimo,  
                                           SueldoMaximo {  
    if (<Condición para detectar sueldo mínimo>)  
        // Lanzar la excepción SueldoMinimo  
    if (<Condición para detectar sueldo máximo>)  
        // Lanzar la excepción SueldoMaximo  
  
    /* Código del método */  
}
```

- En la signatura del método se deberán de indicar **todas las excepciones** que podrían ocurrir durante la ejecución del método, es decir, todas las excepciones que se generarán
- Si las excepciones pertenecen a la misma jerarquía, es decir, tienen una misma clase base, **sería suficiente** con etiquetar el método con la clase base o clases superiores a ella

# Gestión de excepciones: Intentar

- La gestión de las excepciones comienza con la invocación de un método<sub>A</sub> desde otro método<sub>B</sub>, de modo que se puede decir que el **método<sub>A</sub> intenta ejecutar con éxito el método<sub>B</sub>**
  - Es necesario indicar en qué parte del código del método<sub>A</sub> se va **intentar la ejecución con éxito** del método<sub>B</sub>, es decir, se intenta la invocación del método<sub>B</sub> esperando obtener el resultado por el cual tiene lugar dicha invocación



```
try {
```

En el **bloque try** se deberá de invocar **obligatoriamente** al método<sub>B</sub>, aunque lo más deseable es que **todo** el código funcional del método<sub>A</sub> se sitúe dentro de un **único bloque try** para que sea más legible

```
}
```

# Gestión de excepciones: Intentar

- En el código de un puede haber **tantos bloques try como invocaciones** a métodos que generan una excepción, incluso aunque sean varias invocaciones a un mismo método
  - Con un único bloque try se hace **más legible** el código del método

```
/* Código método A */
try {
    /* Invocación a método B1 */
} catch (ExcepcionB1 exc) { /* Tratamiento 1 */ }
/* Continúa código método A */
try {
    /* Invocación a método B2 */
} catch (ExcepcionB2 exc) { /* Tratamiento 2 */ }
/* Continúa código método A */
try {
    /* Invocación a método B1 */
} catch (ExcepcionB1 exc) { /* Tratamiento 1 */ }
/* Continúa código método A */
try {
    /* Invocación a método B3 */
} catch (ExcepcionB3 exc) { /* Tratamiento 3 */ }
```

vs.

```
try {
    /* Código método A */
    /* Invocación a método B1 */
    /* Continúa código método A */
    /* Invocación a método B2 */
    /* Continúa código método A */
    /* Invocación a método B1 */
    /* Continúa código método A */
    /* Invocación a método B3 */
} catch (ExcepcionB1 exc) { /* Tratamiento 1 */ }
} catch (ExcepcionB2 exc) { /* Tratamiento 2 */ }
} catch (ExcepcionB3 exc) { /* Tratamiento 3 */ }
```

El código está **unificado** en un único bloque de código



# Gestión de excepciones: Intentar

```
public static void main(String[] args) {  
    try {  
        Empresa empresa= new Empresa("MiEmpresa");  
        Empleado empleadoB1= new Empleado("EmpleadoB1");  
        Empleado empleadoD1= new Empleado("EmpleadoD1");  
        Empleado empleadoP= new Empleado("EmpleadoP1");  
  
        empresa.getEmpleados().add(empleadoB1);  
        empresa.getEmpleados().add(empleadoD1);  
        empresa.getEmpleados().add(empleadoP);  
  
        for(Empleado empdo : empresa.getEmpleados()) {  
            empdo.setSueldo(empdo.calcularSueldo()/1.05f);  
        }  
    } catch(SueldoMinimo sm) {  
        System.out.println("Error -> " + sm.getMessage());  
    } catch(SueldoMaximo sm) {  
        System.out.println("Error -> " + sm.getMessage());  
    }  
}
```

El método **setSueldo** es susceptible de generar una excepción del tipo **SueldoMinimo**, de modo que **será necesario** colocar las invocaciones de dicho método en un bloque try

Aunque existen tres invocaciones del método, **no es obligatorio** tener un bloque try para cada una de ellas

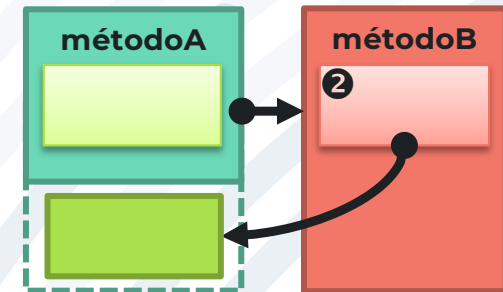
Tampoco es necesario que el resto del código del método se encuentre entre en un único bloque try, pero **simplifica enormemente** el código

Al establecer el sueldo para empleadoB1 se interrumpirá el flujo de ejecución y las siguientes líneas de código del bloque try **no llegarán** a ejecutarse nunca



# Gestión de excepciones: lanzar

- Una vez se invoca la ejecución del método<sub>B</sub>, el control del programa pasa al código de dicho método, que tiene dos partes: **(1) la detección** y **(2) la generación** de la excepción para que sea tratada en el método que realizó la invocación



- (1) La detección de la excepción consiste en especificar dentro del código del método<sub>B</sub> las **condiciones** en las cuales se genera la excepción
  - Chequear si el numerador de la división es 0
  - Chequear los **códigos de error** descritos en el proyecto del Risk, que, en realidad, no son errores, sino **excepciones** que ocurren durante la ejecución de los métodos

# Gestión de excepciones: lanzar

- (2) Al generar la excepción se completa lo que se entiende por **lanzar la excepción**, que consiste en crear un objeto del tipo de excepción que se transferirá al trozo de código en el cual tiene lugar el tratamiento de la excepción
- Cuando se genera la excepción, es decir, **cuando se crea el objeto del tipo excepción**, es habitual incluir en ella **información sobre el contexto** en el que ha ocurrido, incluyendo **(a)** una descripción textual sobre la causa que la ha provocado; y/o **(b)** referencias a objetos que se podrán usar en el tratamiento de la excepción
  - Al lanzar una excepción **se interrumpe la ejecución del método** en el mismo punto en el cual se ha lanzado, es decir, el flujo de programa abandona el método y apunta directamente al código en el que se lleva a cabo el tratamiento de la excepción

# Gestión de excepciones: lanzar

```
if(<condición>) {  
    <tipo_excepción> <nombre_objeto> = new <constructor>;  
    throw <nombre_objeto>;  
}
```

- La palabra **throw** se utiliza para indicar que se interrumpe la ejecución del método y se lanza la excepción para que pueda ser capturada por el código en el que se trata la excepción, que es el **punto en el que continúa** la ejecución del programa
- La información del contexto de ocurrencia de la excepción **se pasa en los argumentos** del constructor de la clase asociada a la excepción

# Gestión de excepciones: lanzar

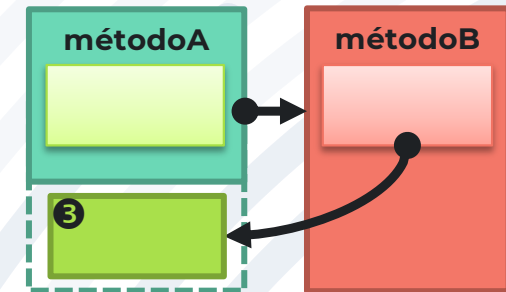
```
public void setSueldo(float sueldo) throws SueldoMinimo, SueldoMaximo {  
    // (1) DETECTAR LA EXCEPCIÓN  
    if(sueldo<950) {  
        // (2) GENERAR LA EXCEPCIÓN "SueldoMinimo"  
        String mensaje= "El sueldo " + sueldo + " es menor que el mínimo";  
        SueldoMinimo error= new SueldoMinimo(mensaje, this);  
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"  
    } else if(sueldo>5000) {  
        // (2) GENERAR LA EXCEPCIÓN "SueldoMaximo"  
        String mensaje= "El sueldo " + sueldo + " es mayor que el máximo";  
        SueldoMaximo error= new SueldoMaximo(mensaje, this);  
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"  
    } else  
        this.sueldo= sueldo;  
}
```

Si **no se cumple** la condición, se ejecuta la operación funcional del método, es decir, asignar un valor al atributo **sueldo**. Esta parte del código está separada de la gestión de la excepción

Si **se cumple** la condición **(1)** se crea el objeto **error** cuyo tipo es la clase de la excepción. Para crear el objeto se usa el constructor de **SueldoMinimo**, al que se le pasa como argumento un texto en el que se indica la causa por la que se generó la excepción; y **(2)** se lanza el objeto **error** mediante **throw**, lo que significa que se interrumpe y se abandona la ejecución del método **setSueldo** en ese punto, de manera que ese objeto se transfiere como entrada al código de tratamiento de la excepción

# Gestión de excepciones: tratar

- Una vez se lanza una excepción, el objeto de tipo excepción se transfiere **como “argumento”** al código en el que se realiza su tratamiento, de modo que dicho código tiene información contextual sobre la ocurrencia de la excepción
- El momento en el que el flujo de programa entra en el código para el tratamiento de la excepción se denomina **captura de la excepción**
  - Típicamente, el tratamiento consistirá en mostrar al usuario y/o grabar en un fichero el **mensaje de texto** que se ha pasado como argumento al constructor en la creación del objeto que se lanza cuando ocurre la condición de la excepción



# Gestión de excepciones: tratar

- El tratamiento de la excepción tiene lugar en los denominados bloques **catch**, que están directamente asociados a los bloques **try**, de modo que un bloque catch **no puede existir** sin haber especificado previamente un bloque try

```
try {  
    /* Código del bloque de intento */  
} catch(<tipo_excepcion> <nombre_excepcion>) {
```

En el **bloque catch** hay código que realiza el tratamiento de la excepción, usando la información de contexto que contiene el objeto <nombre\_objeto> de la clase <tipo\_excepción>

```
}
```

# Gestión de excepciones: tratar

- **Restricciones entre los bloques try-catch**

- Si existe un bloque catch deberá existir necesariamente **un único bloque try**
- Un bloque try deberá tener asociado, **al menos**, un bloque catch
- Un bloque try **podría tener tantos** bloques catch **como** el número de tipos de excepciones se puedan lanzar en el bloque try
- Un bloque try **podría tener menos** bloques catch **que** el número de excepciones que se hayan lanzado en el bloque try
  - Si las excepciones que se lanzan tienen la misma clase base, entonces puede definirse **un único bloque catch**
  - Si el tratamiento es el mismo puede usarse **multi-catch**



# Gestión de excepciones: tratar

```
public static void main(String[] args) {  
    try {  
        Empresa empresa= new Empresa("MiEmpresa");  
        Empleado empleadoB1= new Empleado("EmpleadoB1");  
        Empleado empleadoD1= new Empleado("EmpleadoD1");  
        Empleado empleadoP= new Empleado("EmpleadoP1");  
  
        empresa.getEmpleados().add(empleadoB1);  
        empresa.getEmpleados().add(empleadoD1);  
        empresa.getEmpleados().add(empleadoP);  
  
        for(Empleado empdo : empresa.getEmpleados()) {  
            empdo.setSueldo(empdo.calcularSueldo()/1.05f);  
        }  
    } catch(SueldoMinimo sm) {  
        System.out.println("Error -> " + sm.getMessage());  
    } catch(SueldoMaximo sm) {  
        System.out.println("Error -> " + sm.getMessage());  
    }  
}
```

Los dos **bloques catch** están asociados al tratamiento de cada una de las excepciones que puede lanzar el método **setSueldo: SueldoMinimo y SueldoMaximo**

El tratamiento que se define para las dos excepciones **es idéntico**, si bien el resultado será diferente, puesto que el mensaje también es distinto. En este caso, **se podría usar multi-catch** para simplificar el código



# Gestión de excepciones: tratar

- La opción **multi-catch** no añaden ningún tipo de semántica a la gestión de la excepción, sino que está orientada a **simplificar** el código del bloque catch

*catch(<tipo\_excepción> | <tipo\_excepción> | \* <nombre\_excepción>)*

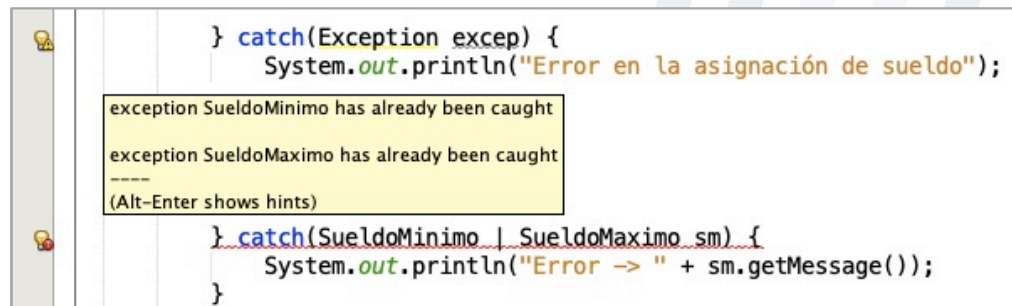
```
} catch(SueldoMinimo sm) {  
    System.out.println("Error -> " + sm.getMessage());  
}  
catch(SueldoMaximo sm) {  
    System.out.println("Error -> " + sm.getMessage());  
}
```

```
} catch(SueldoMinimo | SueldoMaximo sm) {  
    System.out.println("Error -> " + sm.getMessage());  
}
```

Con multi-catch se **reduce** tanto más el código cuantas más excepciones hay que se tratan de la misma forma

# Gestión de excepciones: tratar

- **La herencia y el polimorfismo** juegan un papel importante a la hora de decidir qué bloque catch se ejecutará cuando se lanza una excepción
  - Si el argumento del bloque catch es una de las clases superiores a una clase<sub>A</sub>, después de ese catch **no podrá existir** un catch con la clase<sub>A</sub>, puesto que la excepción se habrá capturado en el catch de la clase superior
    - En esta situación tendrá lugar un **error de compilación**



```
} catch(Exception excep) {  
    System.out.println("Error en la asignación de sueldo");  
}  
  
} catch(SueldoMinimo | SueldoMaximo sm) {  
    System.out.println("Error -> " + sm.getMessage());  
}
```

exception SueldoMinimo has already been caught  
exception SueldoMaximo has already been caught  
----  
(Alt-Enter shows hints)

# Gestión de excepciones: tratar

- En el tratamiento de las excepciones, además de los bloques catch también existe el **bloque finally**, que se ejecuta **siempre**; independientemente de la excepción que se haya capturado o incluso aunque no se haya capturado ninguna excepción
  - Un bloque try puede tener asociado un **único** bloque finally
  - El bloque finally **siempre deberá ir después** de todos los bloques catch asociados al tratamiento de la excepción
  - No es posible definir un bloque finally si no existe, **al menos**, un bloque catch el que se capturen las excepciones generadas en el bloque try
  - Uno de los usos habituales del bloque finally es **liberar recursos**

# Gestión de excepciones: tratar

```
public static void main(String[] args) {
    ArrayList<Empleado> empleadosConSueldo= new ArrayList<>();
    try {
        Empresa empresa= new Empresa("MiEmpresa");
        Empleado empleadoB1= new Empleado("EmpleadoB1");
        Empleado empleadoD1= new Empleado("EmpleadoD1");
        Empleado empleadoP= new Empleado("EmpleadoP1");

        empresa.getEmpleados().add(empleadoB1);
        empresa.getEmpleados().add(empleadoD1);
        empresa.getEmpleados().add(empleadoP);

        for(Empleado empdo : empresa.getEmpleados()) {
            empdo.setSueldo(empdo.calcularSueldo()/1.05f);
        }
    } catch(SueldoMinimo sm) {
        System.out.println("Error -> " + sm.getMessage());
    } catch(SueldoMaximo sm) {
        System.out.println("Error -> " + sm.getMessage());
    } finally {
        System.out.println("Empleados con sueldo");
        for(Empleado empleado : empleadosConSueldo)
            System.out.println(empleado.getNombre());
    }
}
```

El **bloque finally** se ejecuta después de que haya ocurrido una excepción en **setSueldo**

Si se genera una excepción, primero se interrumpirá el bucle for-each en la iteración en la cual ha tenido lugar; después se ejecutará el bloque catch asociado al tipo de excepción que se ha generado; y finalmente se ejecuta el bloque finally, que muestra la lista de los empleados a los que se les ha asignado un sueldo

# Gestión de excepciones: uniendo todo

```
public static void main(String[] args) {
    ArrayList<Empleado> empleadosConSueldo= new ArrayList<>();
    try {
        Empresa empresa= new Empresa("MiEmpresa");
        Empleado empleadoB1= new Empleado("EmpleadoB1");
        Empleado empleadoD1= new Empleado("EmpleadoD1");
        Empleado empleadoP= new Empleado("EmpleadoP1");

        empresa.getEmpleados().add(empleadoB1);
        empresa.getEmpleados().add(empleadoD1);
        empresa.getEmpleados().add(empleadoP);

        for(Empleado empdo : empresa.getEmpleados()) {
            empdo.setSueldo(empdo.calcularSueldo()/1.05f);
        }
    } catch(SueldoMinimo sm) {
        System.out.println("Error -> " + sm.getMessage());
    } catch(SueldoMaximo sm) {
        System.out.println("Error -> " + sm.getMessage());
    } finally {
        System.out.println("Empleados con sueldo");
        for(Empleado empleado : empleadosConSueldo)
            System.out.println(empleado.getNombre());
    }
}
```

```
public void setSueldo(float sueldo) throws SueldoMinimo, SueldoMaximo {
    // (1) DETECTAR LA EXCEPCIÓN
    if(sueldo<950) {
        // (2) GENERAR LA EXCEPCIÓN "SueldoMinimo"
        String mensaje= "El sueldo " + sueldo + " es menor que el mínimo";
        SueldoMinimo error= new SueldoMinimo(mensaje, this);
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"
    } else if(sueldo>5000) {
        // (2) GENERAR LA EXCEPCIÓN "SueldoMaximo"
        String mensaje= "El sueldo " + sueldo + " es mayor que el máximo";
        SueldoMaximo error= new SueldoMaximo(mensaje, this);
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"
    } else
        this.sueldo= sueldo;
}
```

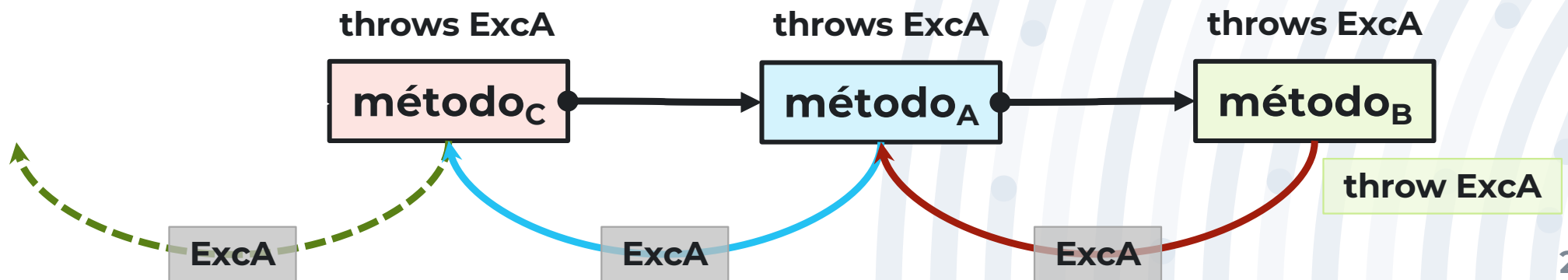
error SueldoMinimo ... #155

sm SueldoMinimo ... #155

El objeto **error** el método setSueldo y el objeto **sm** del bloque catch apuntan a la misma dirección de memoria, es decir, **son el mismo objeto**

# Revisitando generación de excepciones

- Un método<sub>A</sub> que invoca a un método<sub>B</sub> que lanza una excepción **no debe tener necesariamente** un try-catch como parte de su código, es decir, el método<sub>A</sub> no tiene por qué realizar la captura de la excepción
  - El método<sub>A</sub> puede delegar el tratamiento de la excepción a otro método<sub>C</sub> que lo invoque, de modo que existe un encadenamiento lanzamientos de excepciones que tiene como fin la unificación del código en el cual se realiza el tratamiento final de la excepción



# Revisitando generación de excepciones

sm= error

```
public static void main(String[] args) {
    ArrayList<Empleado> empleadosConSueldo= new ArrayList<>();
    try {
        Empresa empresa= new Empresa("MiEmpresa");
        empresa.getEmpleados().add(new Empleado("EmpleadoB1"));
        empresa.getEmpleados().add(new Empleado("EmpleadoD1"));
        empresa.getEmpleados().add(new Empleado("EmpleadoP1"));
        empresa.nuevosSueldos(1.05f);
    } catch (SueldoMinimo | SueldoMaximo sm) {
        System.out.println("Error -> " + sm.getMessage());
    } finally {
        System.out.println("Sueldos actualizados");
    }
}
```

error se genera en **setSueldo**, pero no se trata en el método que lo ha invocado, **nuevosSueldos**, sino en el método **main**

```
public void nuevosSueldos(float factor) throws SueldoMinimo, SueldoMaximo {
    for (Empleado emp : this.empleados)
        emp.setSueldo(emp.calcularSueldo()/factor);
}
```

throw error

```
public void setSueldo(float sueldo) throws SueldoMinimo, SueldoMaximo {
    // (1) DETECTAR LA EXCEPCIÓN
    if(sueldo<950) {
        // (2) GENERAR LA EXCEPCIÓN "SueldoMinimo"
        String mensaje= "El sueldo " + sueldo + " es menor que el mínimo";
        SueldoMinimo error= new SueldoMinimo(mensaje, this);
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"
    } else if(sueldo>5000) {
        // (2) GENERAR LA EXCEPCIÓN "SueldoMaximo"
        String mensaje= "El sueldo " + sueldo + " es mayor que el máximo";
        SueldoMaximo error= new SueldoMaximo(mensaje, this);
        throw error; // LANZAR LA EXCEPCIÓN A TRAVÉS DE "throw"
    } else
        this.sueldo= sueldo;
}
```



# Bibliografía

- ① [https://www.linuxtopia.org/online\\_books/programming\\_books/thinking\\_in\\_java/](https://www.linuxtopia.org/online_books/programming_books/thinking_in_java/)
- ② <https://docs.oracle.com/javase/tutorial/index.html>
- ③ <https://www.baeldung.com/get-started-with-java-series>
- ④ <https://www.geeksforgeeks.org/java/>
- ⑤ <https://www.guru99.com/java-tutorial.html>