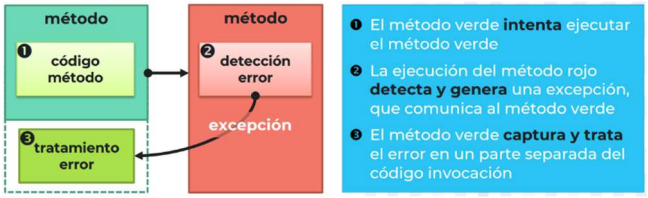
Tema 7: Excepciones

- 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.

Concepto de excepción

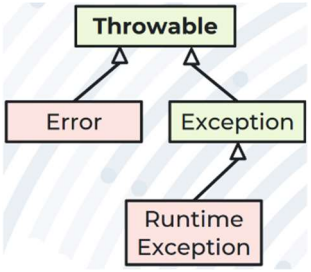
- En la programación de los métodos se debe favorecer la **separación** lógica de los diferentes tipos de código:
  - Funcionalidad provista por el método.
  - Interacción con usuarios u otros programas.
  - Almacenamiento y acceso a los datos.
  - **Gestión de errores**.

- Una EXCEPCIÓN es la ocurrencia de **errores y/o situaciones de interés** durante la **ejecución** de los métodos que proporcionan la funcionalidad del programa.
- El mecanismo de **Java** para el tratamiento de las excepciones aplica la filosofía de **separar el código** para la detección, comunicación y tratamiento de excepciones del código que proporciona la funcionalidad del programa.



Definición de excepción

- Todas las excepciones son una clase derivada de la clase $Throwable$, cuyos métodos más usados son:
  - $String\ getMessage() \rightarrow$ devuelve el mensaje indicado en la ocurrencia de la excepción.
  - $void\ printStackTrace() \rightarrow$ imprime por consola la secuencia de invocaciones de métodos que llevó a la ocurrencia de la excepción.



En Java existen dos tipos de excepciones:

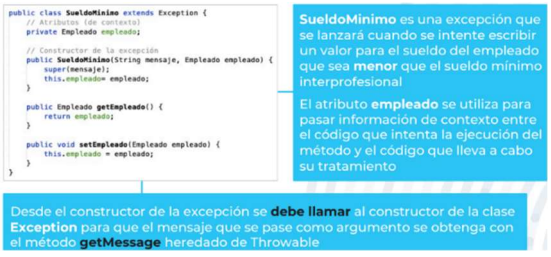
- **Excepciones chequeadas** → se comprueba en tiempo de compilación en qué métodos pueden ocurrir.
  - Los métodos en los que puedan ocurrir una determinada excepción chequeada durante su ejecución deben **indicarlo explícitamente**.
  - Son clases derivadas de la clase $Exception$.
- **Excepciones no chequeadas** → no se comprueba en tiempo de compilación en qué métodos pueden ocurrir.
  - El programador debe chequear en qué métodos puede ocurrir una determinada excepción no chequeada para intentar capturarla y tratarla.
  - Son clases derivadas de las clases $RuntimeException$ y $Error$.
- Si el error o situación de interés que provoca la ocurrencia de la excepción no impide la ejecución del programa, esta debería ser chequeada.

Excepciones de Java chequeadas	Excepciones de Java no chequeadas
IOException	ArithmeticException
SQLException	ClassCastException
URISyntaxException	IndexOutOfBoundsException
...	NullPointerException
	UnsupportedOperationException
	...

- La definición de excepciones como clases derivadas de $Exception$ le da al programador más control sobre su gestión.
  - Estas clases se instancian en el momento en el que ocurre la excepción.
    - Sus constructores deben invocar al constructor $Exception(String\ message)$, en el que $message$ es el mensaje con el que el programador suele explicar por qué ha ocurrido la excepción para mostrárselo al usuario.
  - Los métodos que puedan provocar ocurrencias de excepciones se deben **etiquetar** para indicar explícitamente qué excepciones pueden provocar.
    - Un mismo método puede estar etiquetado con más de una excepción, es decir, más de una subclase de $Exception$.
    - Si todas las excepciones que podría provocar un método pertenecen a la misma jerarquía, sería suficiente etiquetar el método con su clase base o clases superiores a ella.

$$< tipo\_acceso > < tipo\_return > metodo\ (< args >)\ \boldsymbol{throws} < \boldsymbol{excepcion_1} >, < \boldsymbol{excepcion_2} >, \ldots$$

$$< tipo\_acceso >\ clase\ (< args >)\ \boldsymbol{throws} < \boldsymbol{excepcion_1} >, < \boldsymbol{excepcion_2} >, \ldots$$

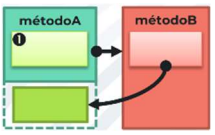


↳ Los constructores también pueden provocar excepciones.

```
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 */
}
```

Gestión de excepciones: intentar



- El bloque $try\{\}$ indica la parte del código en el que un método$_A$ **intenta ejecutar con éxito** un método$_B$ que puede provocar alguna excepción.
- En el código del método$_A$ puede haber tantos bloques $try\{\}$ como invocaciones a métodos que pueden provocar una excepción, incluso aunque sean varias invocaciones al mismo método.
  - ↳ Sin embargo, con un único bloque $try\{\}$ el código es más legible.

```
/* 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

```
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

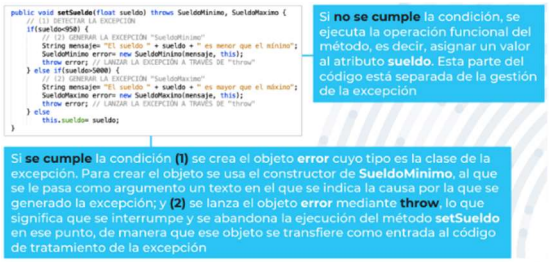
Tampoco es necesario que el resto del código del método se encuentre entre en un único bloque try, pero **simplifica enormemente** el código

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

GESTIÓN DE EXCEPCIONES: LANZAR

- Una vez se invoque desde método$_A$ el método$_B$, comenzará la ejecución de su código, que tendrá 3 partes:
  1. Detección de la excepción → se especifican las condiciones en las cuales se genera la excepción.
  2. Generación de la excepción → si se cumplen esas condiciones, se crea un objeto del tipo de excepción correspondiente.
     - Es habitual pasar información del contexto en el que ocurrió la excepción como argumentos de su constructor, como un mensaje y/o referencias a objetos que se podrán usar en el tratamiento de la excepción.
  3. Lanzamiento de la excepción → el objeto creado se transfiere al trozo de código en el que se trata la excepción usando $throw(excepcion)$.
     - Al ejecutar $throw()$ se interrumpe la ejecución del método$_B$. El flujo del programa continuará en el código que lleva a cabo el tratamiento de la excepción.

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

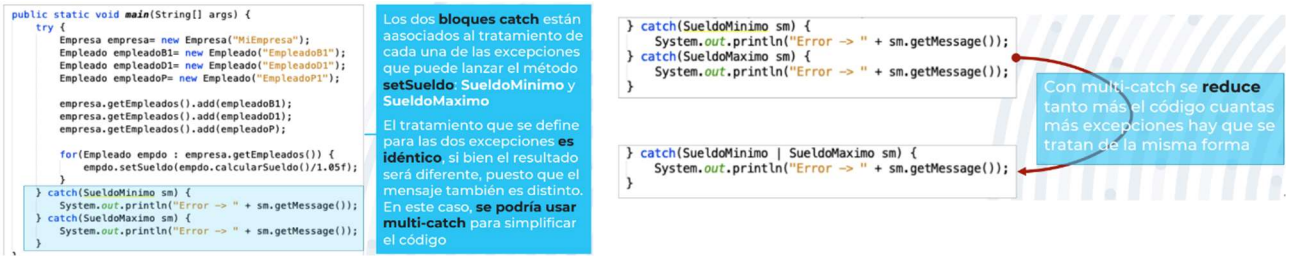


GESTIÓN DE EXCEPCIONES: TRATAR

- El bloque $catch(< tipo > nombre)\{\}$ se encarga de **capturar el objeto excepción** lanzado por el método$_B$ y contiene el código en el que se **trata la excepción**, el cual suele consistir en usar $getMessage()$ para mostrar al usuario y/o grabar en un archivo el mensaje que se pasó como argumento al constructor de la excepción.
  - ↳ El bloque $catch()\{\}$ puede estar en el método$_A$ o en cualquier otro método de la secuencia de invocaciones que llevó a la ocurrencia de la excepción.
- Los bloques $catch()\{\}$ están directamente asociados a los bloques $try\{\}$, con las siguientes restricciones:
  - Un bloque $catch()\{\}$ tiene que estar asociado a un único bloque $try\{\}$.
  - Un bloque $try\{\}$ tiene que estar asociado, al menos, a un bloque $catch()\{\}$.
  - Un bloque $try\{\}$ podría estar asociado a tantos bloques $catch()\{\}$ como excepciones distintas se puedan lanzar en él.
  - Un bloque $try\{\}$ podría estar asociado a menos bloques $catch()\{\}$ que excepciones distintas se puedan lanzar en él.
    - Si varias de estas excepciones tienen la misma clase base, entonces puede definirse un único bloque $catch()\{\}$ para todas ellas con su clase base.
    - Si varias de estas excepciones tienen el mismo tratamiento, entonces se puede usar **multi-catch:**

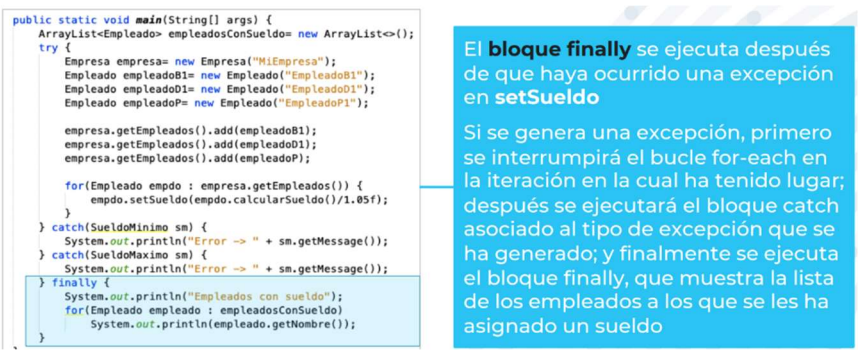
      $$catch(< excepcion_1 > | < excepcion_2 > | \ldots)\{\ldots tratamiento \ldots\}$$

      ↳ El multi-catch no añade ninguna semántica a la gestión de las excepciones, simplemente simplifica el código del bloque $catch\{\}$.



- La **herencia** y **polimorfismo** juegan un papel importante a la hora de decidir qué bloque $catch()\{\}$ se ejecutará cuando el método$_B$ lance una excepción:
  Si el argumento del bloque $catch()\{\}$ es una clase superior a una clase A, después de él no podrá existir otro cuyo argumento sea la clase A o clases inferiores, pues la excepción ya se habrá capturado en el primero. Añadirlo provocaría un **error de compilación**.

- En el tratamiento de las excepciones, además de los bloques $catch()\{\}$ también existe un bloque $finally\{\}$ que se ejecuta siempre, independientemente de la excepción que se haya capturado o incluso si no se ha capturado ninguna.
  - Un bloque $finally\{\}$ tiene que estar asociado a un único bloque $try\{\}$.
  - Un bloque $try\{\}$ puede estar asociado a un único bloque $finally\{\}$.
  - Un bloque $finally\{\}$ siempre va después de todos los bloques $catch()\{\}$ asociados al $try\{\}$.
- El bloque $finally\{\}$ suele usarse para liberar recursos (por ejemplo, cerrar archivos, finalizar una conexión con una base de datos, etc.).



GESTIÓN DE EXCEPCIONES: PROCESO COMPLETO

```java
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());
    }
}
```

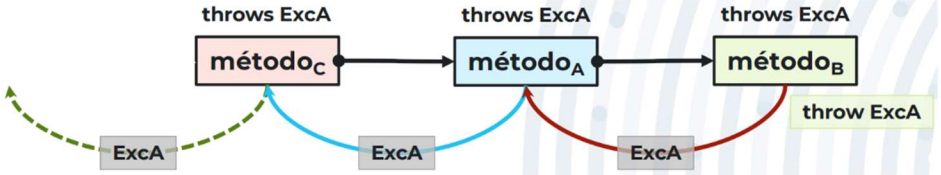
```java
public void setSueldo(float sueldo) throws SueldoMinimo, SueldoMaximo {
    // (1) DETECTAR LA EXCEPCIÓN
    if(sueldo<950) {
        // (2) GENERAR LA EXCEPCIÓN "SueldoMinimo"
        String mensaje= "El sueldo " + sueldo + " es menor que el mínimo";
        SueldoMinimo error= new SueldoMinimo(mensaje, 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**

- El método$_A$ no tiene por qué tener un bloque $try\{\} - catch()\{\}$ en el que gestione las excepciones del método$_B$: puede **delegar** su captura y tratamiento a otro método que lo invoque indicando que él mismo provoca las mismas excepciones que el método$_B$.
- Así, se pueden **encadenar lanzamientos** de excepciones para unificar el código del tratamiento.



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