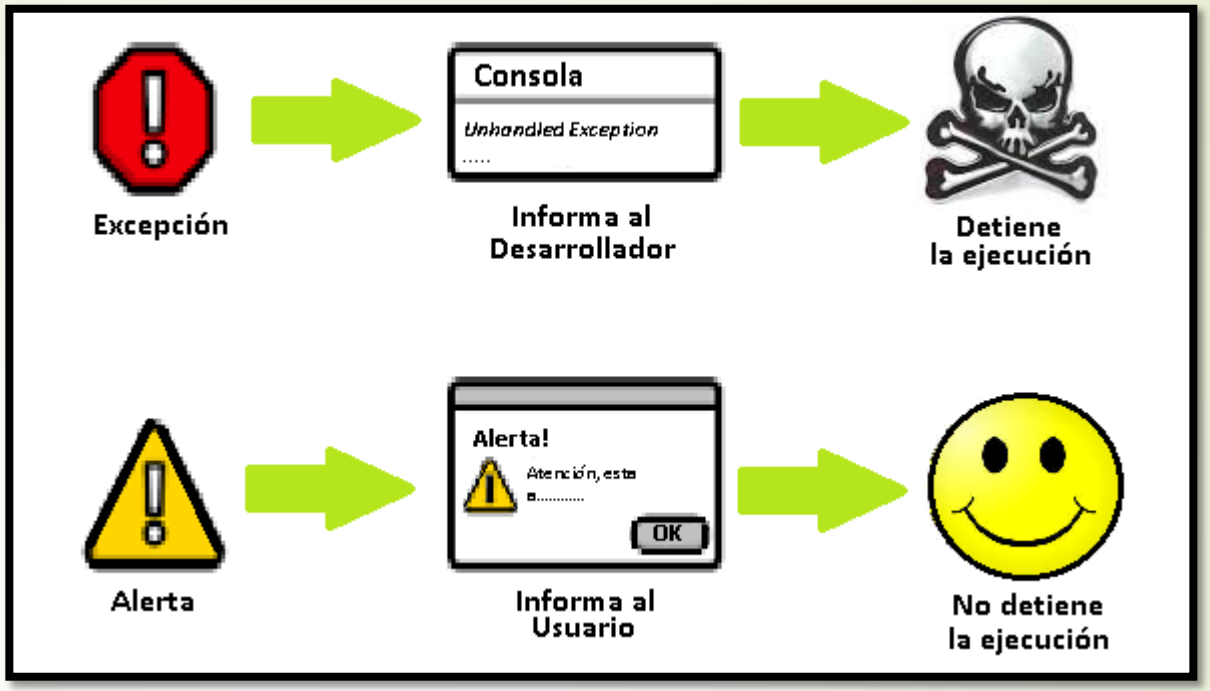
Desarrollo de Aplicaciones Web y Multiplataforma: **Programación**

DOCENTE: Daniel López Lozano

escuelaartegranada



Tema 8.
Gestión de Excepciones

Índice de contenidos

- Durante la ejecución de un programa se pueden producir errores con diversos niveles de severidad:
  - **Un índice fuera de rango.**
  - **Una división por cero.**
  - **Un fichero que no existe.**
  - **Una conexión de red que falla.**
  - **etc.**
- El código sería poco legible si continuamente tuviésemos que comprobar las posibles condiciones de error sentencia tras sentencia.

- Consideremos el siguiente código del método que lee un fichero y copia su contenido en memoria.

```
void leerFichero() {
    abrir el fichero;                    // ¿Qué pasa si el fichero no puede abrirse?
    determinar la longitud del fichero; // ¿Qué pasa si no es posible?
    reservar la memoria suficiente;      // ¿Qué pasa si no hay memoria suficiente?
    copiar el fichero en memoria;        // ¿Qué pasa si falla la lectura?
    cerrar el fichero;                   // ¿Qué pasa si el fichero no puede cerrarse?
}
```

- El control de errores hace que el código sea difícil de leer y modificar. Se pierde el flujo lógico de ejecución.

```
TipoDeCodigoDeError leerFichero() {
    TipoDeCodigoDeError codigoDeError = 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) {
                    codigoDeError = -1;
                }
            } else {
                codigoDeError = -2;
            }
        } else {
            codigoDeError = -3;
        }
```

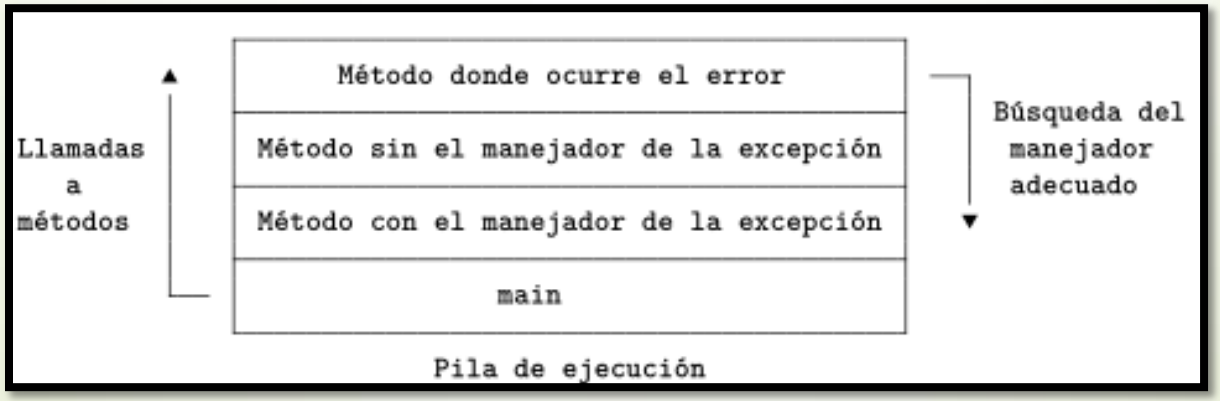
- El mecanismo de las excepciones proporciona una forma clara de gestionar posibles errores sin oscurecer el código.
- No nos liberan de hacer la detección, de informar y de manejar los errores.
- Pero nos permiten escribir el flujo principal de nuestro código en un sitio y de tratar los casos excepcionales separadamente.

Esquema de código usando 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 el cierre del fichero) {
        // ...
    }
}
```

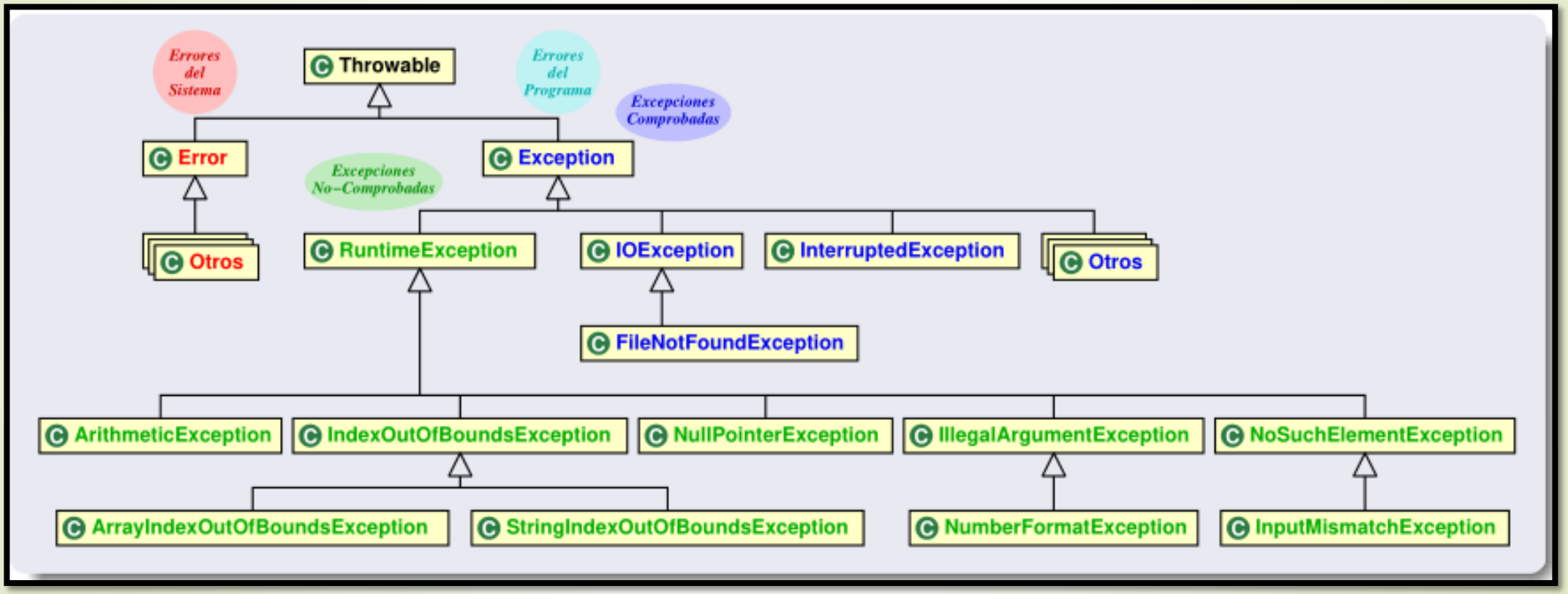
- Una excepción es un evento (una señal) que interrumpe el flujo normal de instrucciones durante la ejecución de un programa como consecuencia de un error (condición excepcional).
- Cuando ocurre un error en un método, entonces se crea un objeto excepción y lo lanza (throw) al sistema de ejecución.
- 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 adecuado que maneje la excepción (manejador de excepción).
- Si el sistema no encuentra un manejador adecuado, entonces el programa termina (mostrando información detallada sobre ella).



- A lo largo de curso hemos visto distinto tipo de excepciones que han ido sucediendo:
  - **IlegalArgumentException**: Error en la llamada de un método por un parámetro erróneo.
  - **InputMismatchException**: Error al introducir datos que no se ajustan al tipo de datos esperado
  - **IndexOutOfBounds**: Error al acceder a una posición no válida de un array.
  - **NullPointerException**: Error al acceder a un objeto null (no inicializado).
  - **NumberFormatException**: Error al convertir una cadena a número.
  - **ArithmeticException**: Error matemático (Ejemplo: dividir por 0).
  - **IOException**: Error genérico de E/S
  - **EOFException**: Error por final inesperado de archivo.
  - **FileNotFoundException**: Error por archivo no encontrado.
  - **Exception**: Engloba a todos las excepciones incluidas las definidas por el programados

Las excepciones forman un jerarquía de herencia



Hay tres puntos de vista para las excepciones:

- **El que maneja (captura) la excepción:** Durante la ejecución de un método, recibe una excepción, y sí sabe cómo resolver la situación que provoca un determinado tipo de excepción, dispone de un tratamiento para ella y recupera el error para poder completar su tarea.
- **El que propaga la excepción:** Durante la ejecución de un método, recibe una excepción, no sabe como tratarla y recuperar el error, por lo que el método no puede completar su tarea, y por lo tanto, propaga la excepción , para informar al nivel superior de la situación anómala y de que la tarea no pudo completarse.

- **El que lanza (eleva) la excepción:** Durante la ejecución de un método, se encuentra una situación anómala que no sabe cómo resolver e impide que el método cumpla su tarea, por lo tanto crea una excepción y la lanza , para informar al nivel superior de la situación anómala y de que la tarea no pudo completarse.

```
try {
//codeblock
} catch(){}
```

Gestión de Excepciones

- La gestión de las excepciones se realiza mediante los bloques try, catch.

```
try {
    //código que potencialmente puede producir una excepción
}catch (tipo_excepcion e1){
    //código para tratar la excepción e1
}catch (tipo_excepcion e2){
    //código para tratar la exepción e2
}
```

- En el bloque try se encapsula todo el código susceptible de producir alguna excepción sea la que sea, incluso si puede generar varias excepciones de tipos diferentes.
- Se define un bloque catch por cada excepción diferente que se pretende tratar, empezando por la más específica y acabando por la más genérica.
- Si un mismo error se puede tratar en más de un catch se tratará con el primero de ellos.
- Si no se produce ningún error no se ejecutará ningún catch

Ejemplo simple de captura de excepciones

```
public double media(int[] datos) {
    //puede que el array esté vacio
    double suma = 0;
    double res;
    for (int i = 0; i < datos.length; ++i) {
        suma += datos[i];
    }
    try{
        res=suma/datos.length;
    }catch(ArithmeticException e){
        System.out.println("Divides por cero");
    }

    return res;
}
```

Ejemplo completo de captura de excepciones

```
public void leeCaracteresFichero() {
    try{
        FileReader br = new FileReader("c:/datos.txt");
        String s=br.readLine();
        ...
    }catch(FileNotFoundException fne){
        sout("error fichero no encontrado");
    }catch(IOExcepcition ioe){
        sout("error al acceder al fichero");
    }catch(Exception e){
        sout("ocurrió un error desconocido");
    }
    f.close();
}
```

- Algunas utilidades interesantes para mostrar información del error o demás datos de la misma son:
- **void printStackTrace():** Este método se puede utilizar dentro de cualquier bloque catch y muestra por pantalla la pila (stack) de llamadas incluyendo los números de línea y ficheros donde se ha producido la excepción. Es decir, la información completa de la excepción.
- **String getMessage():** Este método se puede utilizar dentro de cualquier bloque catch y devuelve una cadena de caracteres con la descripción de la excepción.

Propagación de Excepciones

- Cuando decidimos propagar una excepción para que sea tratada en métodos superiores, es necesario indicarlo en la cabecera del método y que :

```
//Exception o cualquier otra comprobada
public void metodo() throws Exception {
    //código del método
}
```

- Las excepción tiene que ser comprobada.
- Cuando un método puede producir más de una excepción, hay que indicarlas todas o indicar una excepción más general que las englobe a todas

Ejemplo simple de propagación de excepciones

```
public void leeCaracteresFichero() throws FileReader,IOExcepcition,Exception{
    FileReader br = new FileReader("c:/datos.txt");
    String s=br.readLine();
    ...
    f.close();
}

//En el main u quien llame a este metodo

    try{
        leeCaracteresFichero();
    }catch(FileNotFoundException fne){
        sout("error fichero no encontrado");
    }catch(IOExcepcition ioe){
        sout("error al acceder al fichero");
    }catch(Exception e){
        sout("ocurrió un error desconocido");
    }
    f.close();
```

Propagar VS Gestionar

- Cuando se produce una excepción tenemos dos alternativas, tratarla en ese lugar, o lanzarla para que sea tratada en un método superior.
- En general, una excepción se propaga si quiere tratarse de forma genérica en otro método superior (posiblemente el main) ya que puede producirse repetidas veces.
- Si queremos un tratamiento más especifico de las excepciones o pretendemos que el funcionamiento de nuestra clase sea más autónomo para que cuando se reutilice no haya que estar pendiente de estas excepciones, trataremos las excepciones donde se producen.

Lanzamiento de Excepciones

- En el código de nuestro método, podemos lanzar voluntariamente excepciones:

```
public void metodo() throws Exception {
    throw new Exception("Mensaje de excepción");
}

//O cualquier otra clase de excepcion
```

- El lanzamiento de excepciones, tiene sentido cuando podemos prever que la excepción va a ocurrir y no queremos tratarla con un simple mensaje de error sino dar la opción de tratarla al programador usando cualquier otro sistema de entrada salida

Lanzamiento de Excepciones

- También es posible definir nuestras propias excepciones:
- Esto tiene el interés de poder hacer gestión de excepciones sobre nuestros propios código cuando no es un error clasificable en una excepción del sistema.
- Como por ejemplo el tipo de Cerveza tiene que ser rubia, tostada o roja por ejemplo.
- Lo anterior siempre lo hemos tratado con un mensaje de error por terminal y lo mejor la aplicación es web y eso no sirve de nada
- Siempre todo se verá mejor con la práctica.

Lanzamiento de Excepciones

- La sintaxis es la siguiente:

```
public class CervezaExcepcion extends Exception{
    public CervezaExcepcion(String mensaje){
        super(mensaje);
    }
}
```

- En este caso la excepción que estamos creando serie comprobado de tipo obligatorio.
- Por lo tanto es obligatorio gestionarla o propagarla.

Lanzamiento de Excepciones

```
public Cerveza(String id, String nombre, char tipo,
               boolean artesanal, double precio,
               int existencias) throws CervezaExcepcion{
    if(tipo=='r' || tipo=='t' || tipo=='n' || tipo=='R'){
        this.id = id;
        this.nombre = nombre;
        this.tipo = tipo;
        this.artesanal = artesanal;
        this.precio = precio;
        this.existencias = existencias;
    }else{
        throw new CervezaExcepcion("TIPO DE CERVEZA INCORRECTO");
    }
}
```

```
System.out.println("Dime el tipo (r,R,n,t):");
tipo=teclado.next().charAt(0);
System.out.println("Dime si es artesanal (true o false):");
artesanal=teclado.nextBoolean();
System.out.println("Dime el precio:");
precio=teclado.nextDouble();
System.out.println("Dime las existencias:");
existencias=teclado.nextInt();
try{
    Cerveza nueva=new Cerveza(id, nombre, tipo, artesanal, precio,
}catch(CervezaExcepcion ce){
    System.out.println(ce.getMessage());
    //o tambien
    JOptionPane.showMessageDialog(null, ce.getMessage());
}
```

Lanzamiento de Excepciones

- En este caso otro caso la excepción que estamos creando serie no comprobada obligatoriamente.
- Por lo tanto no es obligatorio gestionarla o propagarla.

```
public class CervezaExcepcion extends RuntimeException{
    public CervezaExcepcion(String mensaje){
        super(mensaje);
    }
}
```

Bibliografía

- **García de Jalón, j.: "Aprende Java como si estuvieras en primero".** Editorial TECNUN. 2000
- **Holzner, S.: "La biblia de JAVA 2".** Editorial Anaya Multimedia 2000.
- **Moreno Pérez, J.C.: "C.F.G.S Entornos de desarrollo"** Editorial RA-MA. 2012
- **Wikipedia, la enciclopedia libre. http://es.wikipedia.org/** Última visita: Octubre 2018.
- **López, J.C.: "Curso de JAVA http://www.cursodejava.com.mx** Última visita: Octubre 2018**.**
- **Documentación oficial Java JSE 8 http://docs.oracle.com/javase/8/** Última visita: Octubre 2015.
- **Programación en castellano: Java. http://www.programacion.net/java Última visita: Octubre 2015.**
- **Benjumea, Vicente: Programación orientada a objetos. http://www.uma.es Ultima visita: Mayo 2020**