

# Gestión de errores mediante Excepciones

Programació Orientada a Objectes

## Controlando errores en el pasado (años 70)...



```
Cliente c = controlador.getClient(dni);
if(c == null) {
    System.err.println("Cliente no encontrado: " + dni);
    return;
}
Factura f = c.getFactura(ref);
if(f == null) {
    System.err.println("Factura no encontrada: " + ref);
    return;
}
Pedido p = f.getPedido(producto);
if(p == null) {
    System.err.println("No hay pedidos para dicho producto");
    return;
}
p.setCantidad(4);
```

4 líneas de código para hacer cosas útiles  
12 líneas para gestionar errores

# ¿No sería más cómodo algo así?

```
// Asumimos que todo es correcto
Cliente c = controlador.getCliente(dni);
Factura f = c.getFactura(ref);
Pedido p = f.getPedido(producto);
p.setCantidad(4);
```

```
Si algo ha ido mal {
    Gestionar el error
}
```

## Solución: Excepciones

- Una excepción es un objeto que contiene ciertos datos que pueden describir un error o situación "anómala" en el programa
  - Tipo de error
  - Mensaje descriptivo
  - Punto del programa donde sucedió el error
- Cuando un método detecta un error, instancia una excepción y la lanza (**throw**).
- En vez de hacer comprobaciones en cada método susceptible de devolver un error, el programador habilita una o varias partes del código que puedan cazar (**catch**) esa excepción y tratar el error.

# Estructura básica

- Lanzando una excepción

```
throw new Exception();
```

ó

```
throw new Exception("Mensaje de error");
```

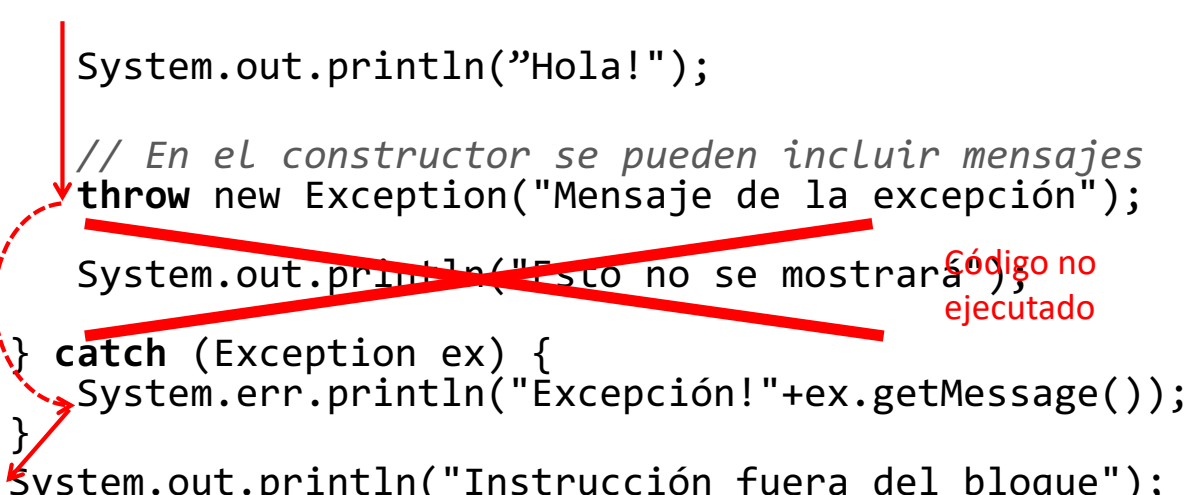
- "Cazando" excepciones

```
try {  
    // código que puede lanzar una excepción.
```

```
} catch(Exception objetoExcepcion) {  
    // código que se ejecuta solo si se lanza  
    // una excepción  
}
```

## Flujo básico del programa

```
try {  
    System.out.println("Hola!");  
    // En el constructor se pueden incluir mensajes  
    throw new Exception("Mensaje de la excepción");  
    System.out.println("Esto no se mostrará");  
} catch (Exception ex) {  
    System.err.println("Excepción!" + ex.getMessage());  
}  
System.out.println("Instrucción fuera del bloque");
```



- Salida por pantalla:

Hola!

Excepción! Mensaje de la excepción

Instrucción fuera del bloque

# Cazando excepciones lanzadas por otros métodos

- Métodos que lanzan (throws) excepciones

```
public float dividir(float a, float b) throws Exception {  
    if(b == 0) {  
        throw new Exception("División por 0!");  
    }  
    return a/b;  
}
```

```
public float raizCuadrada(float numero) throws Exception  
{  
    if(numero < 0) {  
        throw new Exception("Raíz de número negativo!");  
    }  
    return Math.sqrt(numero);  
}
```

# Cazando excepciones lanzadas por otros métodos

- Código que recoge (catch) excepciones

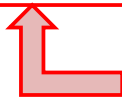
```
public void calcularCosas() {  
    float a,b;  
    try {  
        System.out.println("Iniciando operaciones...");  
  
        a = raizCuadrada(9);  
        b = divide(a,0);  
  
        System.out.println("Fin de las operaciones");  
    } catch(Exception ex) {  
        System.err.println("Sucedió un error: "  
                               + ex.getMessage());  
        ex.printStackTrace();  
    }  
}
```

# Sintaxis de las excepciones

- El código anterior mostraría algo similar a esto en pantalla:  
Iniciando operaciones...

Sucedió un error: Division por 0!

```
java.lang.Exception: Division por 0!  
  at moo.Calculadora.dividir(Calculadora.java:14)  
  at moo.Calculadora.calcularCosas(Calculadora.java:28)  
  at moo.Calculadora.main(Calculadora.java:42)
```



Stack Trace. Muestra la siguiente información:

1. Tipo de excepción: Mensaje de la excepción.
2. Método, clase y línea donde sucedió la excepción.
3. Método, clase y línea que llamó al método anterior.
4. Método, clase y línea que llamó al método anterior.
5. ....
6. Método, clase y línea que llamó al método anterior.

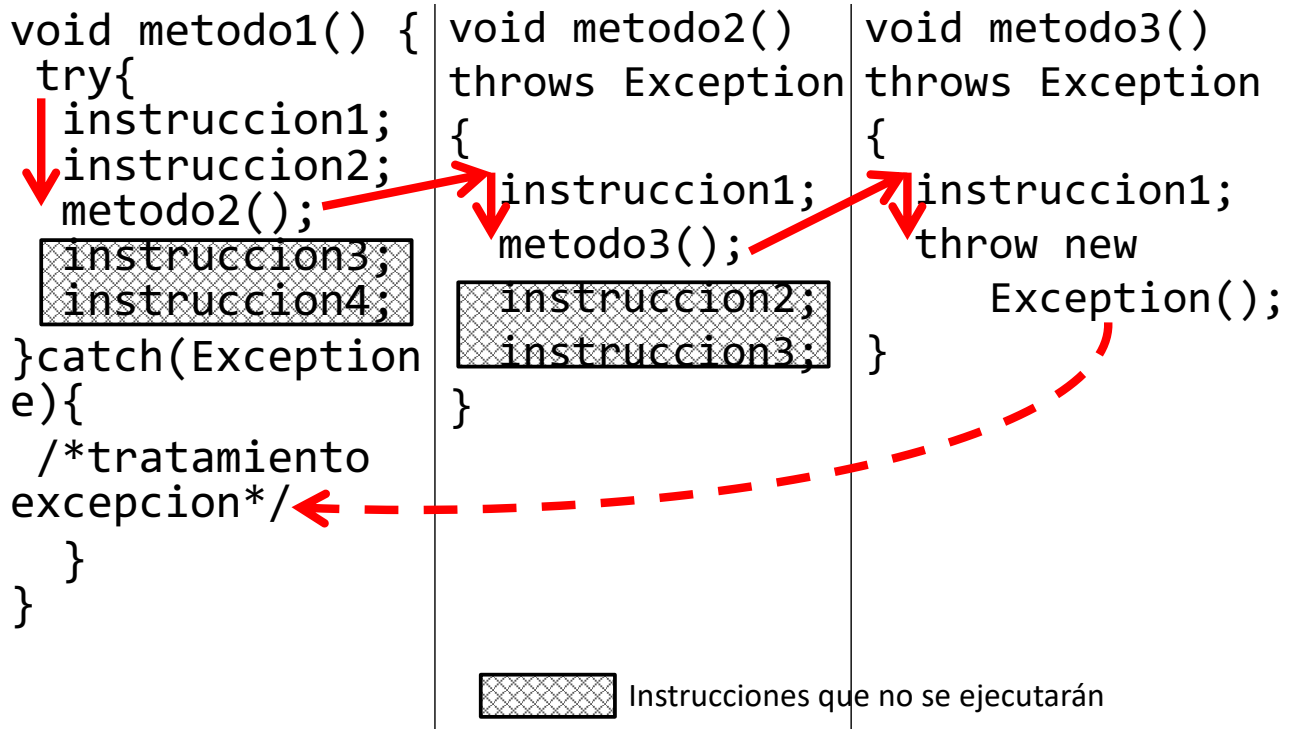
## Relanzamiento de excepciones

- Cuando un método no "caza" una excepción, la relanza al método que lo llamó.

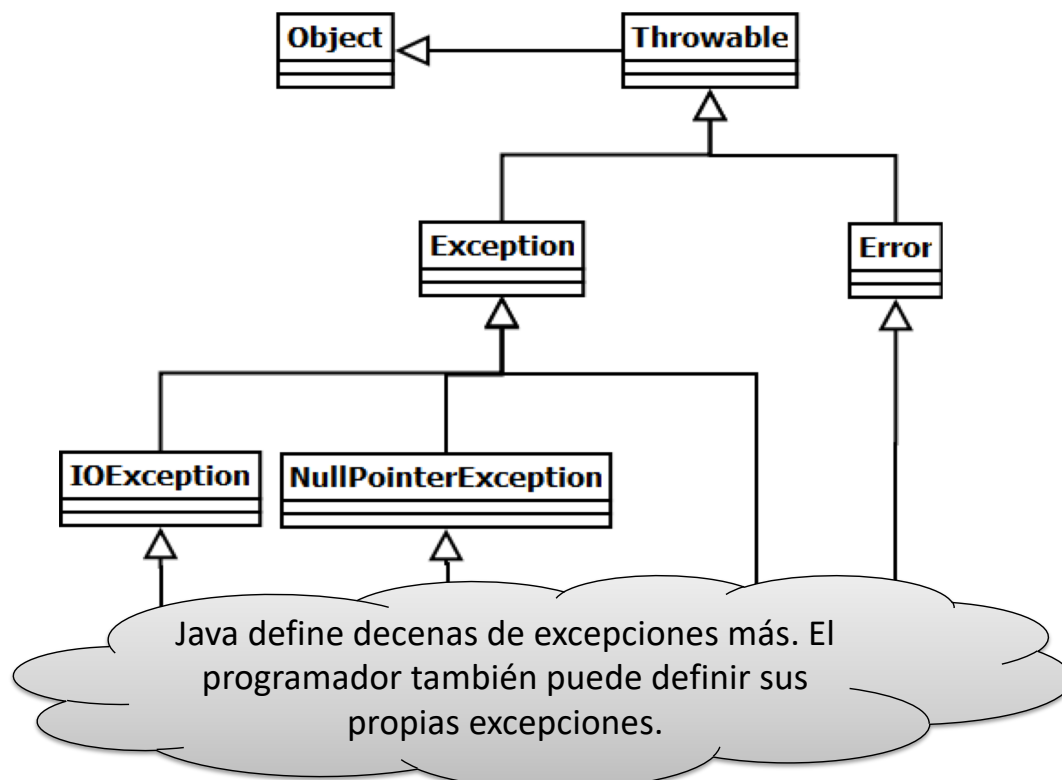
```
public void A() throws Exception {  
    throw new Exception("ERROR!");  
}  
public void B() throws Exception {  
    A();  
}  
public void C() {  
    try{  
        B();  
    } catch(Exception ex) {  
        ex.printStackTrace();  
    }  
}
```

A pesar de que B() no lanza directamente ninguna excepción, no recoge las de A() y la relanza. C() la recoge. Tenemos que poner el "throws"

# Flujo del programa



## Jerarquía de Excepciones



# Definiendo nuestras propias excepciones

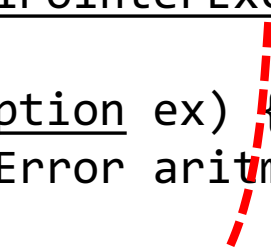
```
public class UsuarioInexistente
                                extends Exception {
    private String usuario;

    public UsuarioInexistente(String usuario) {
        super("El usuario " + usuario
              + " no existe");
        this.usuario = usuario;
    }
    public String getUsuario() {
        return usuario;
    }
}
```

## Gestionando múltiples excepciones

- El siguiente código **no** cazará la excepción.

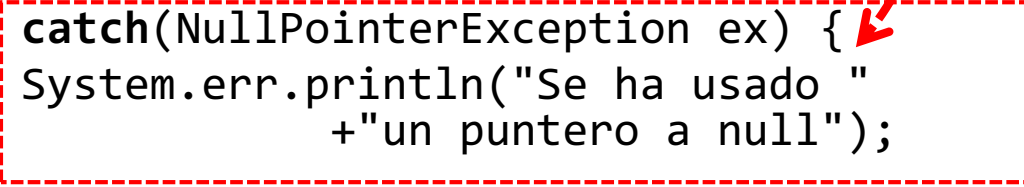
```
String str = null;
try {
    if(str==null)
        throw new NullPointerException();
    str.toUpperCase();
} catch(ArithmeticException ex) {
    System.err.println("Error aritmético");
}
```



# Gestionando múltiples excepciones

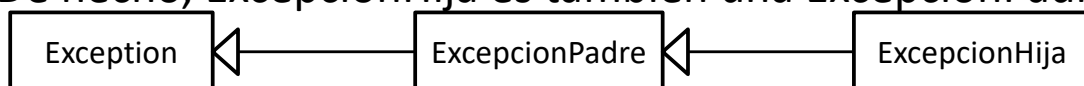
- El siguiente código sí cazará la excepción

```
String str = null;
try {
    if(str==null)
        throw new NullPointerException();
    str.toUpperCase();
} catch(ArithmeticException ex) {
    System.err.println("Error aritmético");
} catch(NullPointerException ex) {
    System.err.println("Se ha usado "
        +"un puntero a null");
}
```

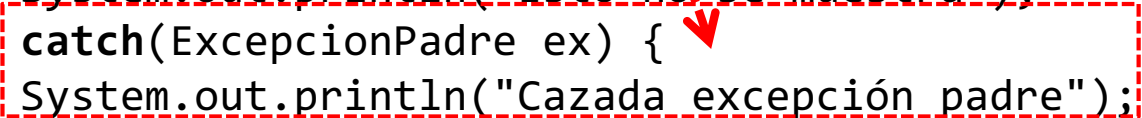


# Gestionando múltiples excepciones

- El bloque que caza la excepción padre cazaría también las excepciones hijas.
- De hecho, ExcepcionHija es también una ExcepcionPadre



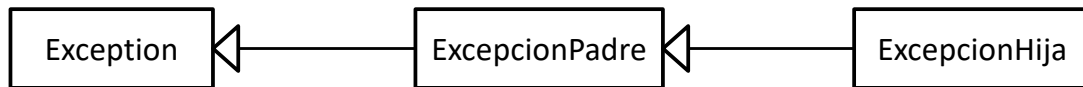
```
try {
    System.out.println("prueba");
    throw new ExcepcionHija();
    System.out.println("Esto no se muestra");
} catch(ExcepcionPadre ex) {
    System.out.println("Cazada excepción padre");
}
```





# Gestionando múltiples excepciones

- Sin embargo, este bloque no cazará la excepción
- ExcepciónPadre no es un tipo de ExcepciónHija.



```
try {  
    System.out.println("prueba");  
    throw new ExcepcionPadre();  
    System.out.println("Esto no se muestra");  
} catch (ExcepcionHija ex) {  
    System.out.println("Cazada excepción"  
                        + e.toString());  
}
```

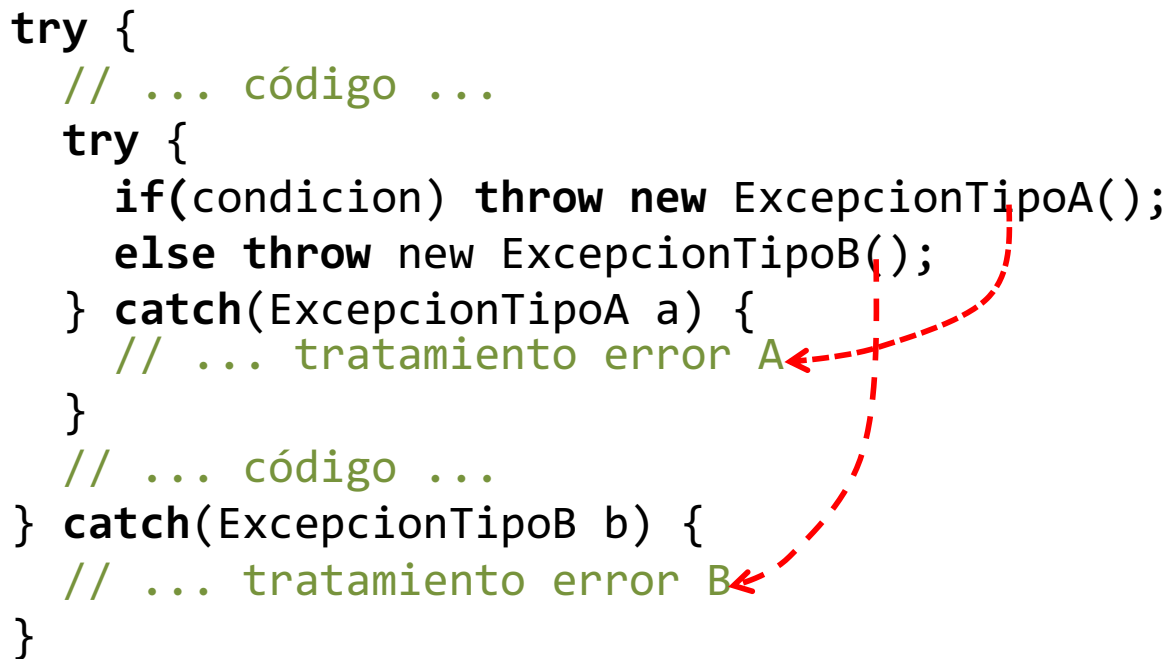
Una flecha roja punteada indica que la excepción lanzada es de tipo ExcepcionPadre, pero el bloque catch solo captura excepciones de tipo ExcepcionHija, por lo que no se ejecutará.

## Múltiples bloques catch

```
try {  
    if(condición) {  
        throw new ExcepcionTipoA();  
    } else {  
        throw new ExcepcionTipoB();  
    }  
} catch (ExcepcionTipoA ex) {  
    //Si condición==true la ejecución viene a parar aquí  
    System.out.println("Cazada excepción del tipo A");  
} catch (ExcepcionTipoB ex) {  
    //Si condición==false la ejecución continúa por aquí  
    System.out.println("Cazada excepción del tipo B");  
}
```

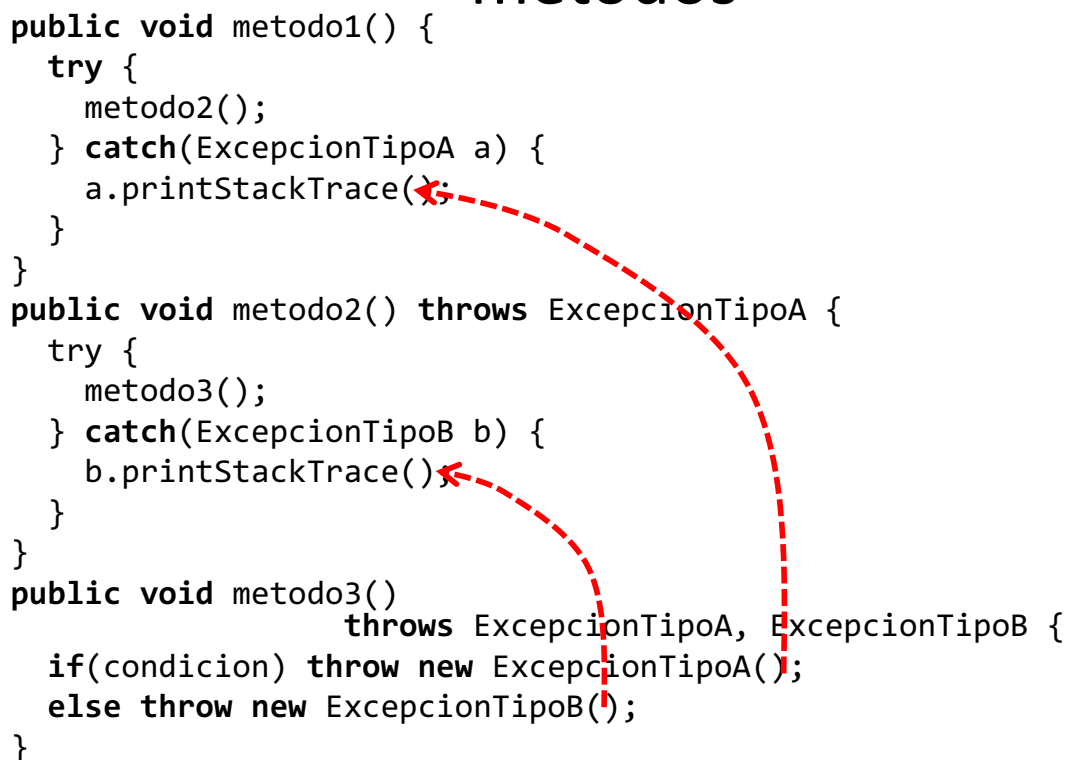
# Excepciones anidadas

```
try {  
    // ... código ...  
    try {  
        if(condicion) throw new ExcepcionTipoA();  
        else throw new ExcepcionTipoB();  
    } catch(ExcepcionTipoA a) {  
        // ... tratamiento error A  
    }  
    // ... código ...  
} catch(ExcepcionTipoB b) {  
    // ... tratamiento error B  
}
```



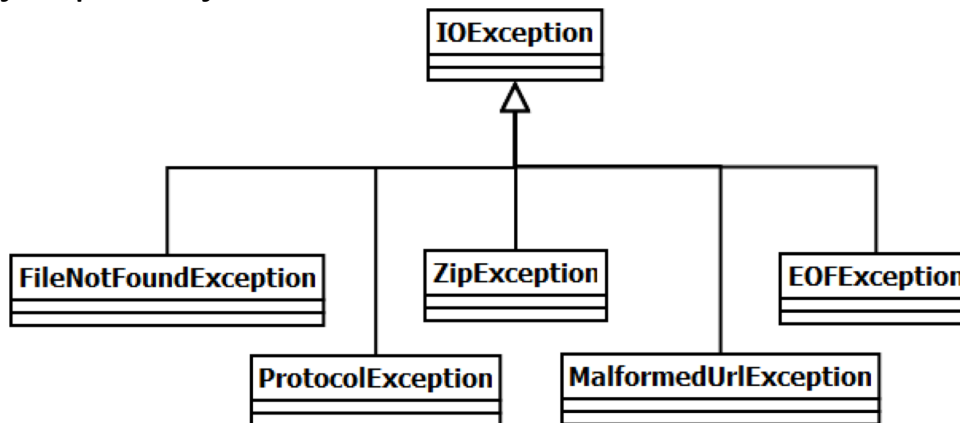
## Excepciones anidadas dentro de métodos

```
public void metodo1() {  
    try {  
        metodo2();  
    } catch(ExcepcionTipoA a) {  
        a.printStackTrace();  
    }  
}  
public void metodo2() throws ExcepcionTipoA {  
    try {  
        metodo3();  
    } catch(ExcepcionTipoB b) {  
        b.printStackTrace();  
    }  
}  
public void metodo3()  
        throws ExcepcionTipoA, ExcepcionTipoB {  
    if(condicion) throw new ExcepcionTipoA();  
    else throw new ExcepcionTipoB();  
}
```

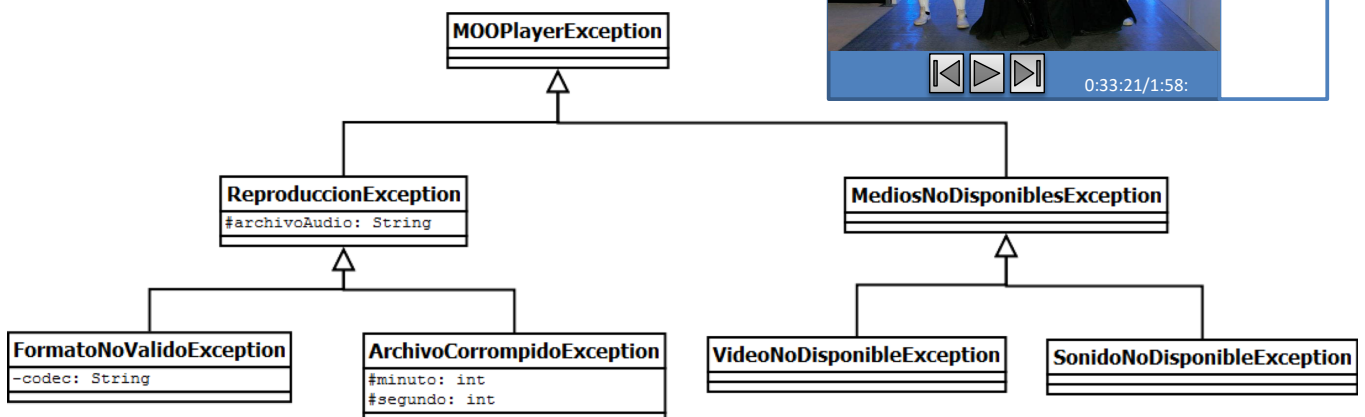
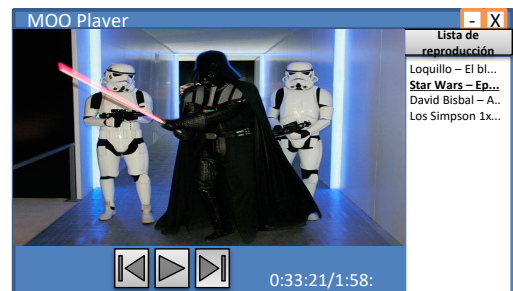


# Criterios de diseño de Excepciones

- Un `throw new Exception("mensaje de error")` apenas da información al programa sobre un error.
- Diseños complejos requieren de múltiples tipos de excepciones. Jerarquizándolas además, podremos tratarlas de manera general o al detalle según las necesidades de cada punto del programa.
- Ejemplo de java:



## Ejemplo: reproductor multimedia



# RuntimeException

- Java detecta ciertos errores en tiempo de ejecución, y lanza automáticamente excepciones. Estas excepciones derivan de RuntimeException, y no necesitan ser declaradas explícitamente en la cabecera de ninguna función.
- Ejemplo: **ArrayIndexOutOfBoundsException**

```
public void metodo1() {  
    int array[] = new array[10];  
    array[15] = 3; //Excepción!!  
}
```

  - No sería práctico hacer bloques try catch cada vez que usamos un simple array!!
- No obstante, en algún momento del programa deberemos evitarlas o "cazarlas".

## RuntimeException: Algunas subclasses

- **ArithmeticException:** Se lanza cuando se intenta hacer una operación no válida (p.ej. dividir por cero)
- **ArrayStoreException:** Se intenta guardar en un array objetos que no son del tipo del array.

```
String cadenas[] = new String[3];  
cadenas[0] = new Integer(6);
```
- **ClassCastException:** Se intenta hacer un *casting* de una referencia a otra de un tipo incorrecto.

```
Figura fig = new Circulo();  
Circulo cir = (Circulo) fig; //Correcto  
Cuadro cuad = (Cuadro) fig; //Lanza ClassCastException
```
- **NullPointerException:** Se intenta operar con una referencia que apunta a null.

```
Object obj = null;  
System.out.println("valor = " + obj.toString());
```