EXCEPCIONES

Las excepciones en Java están destinadas, al igual que en el resto de los lenguajes que las soportan, para la detección y corrección de errores. Si hay un error, la aplicación no debería morirse y generar un *core* (o un *crash* en caso del DOS). Se debería lanzar (*throw*) una excepción que nosotros deberíamos capturar (*catch*) y resolver la situación de error. Java sigue el mismo modelo de excepciones que se utiliza en C++. Utilizadas en forma adecuada, las excepciones aumentan en gran medida la robustez de las aplicaciones.

MANEJO DE EXCEPCIONES

Vamos a mostrar como se utilizan las excepciones, reconvirtiendo nuestro arreglo de saludo a partir de un bucle:

```
public class EjExcepciones {

    public static void main(String[] args) {

       int[] a = {5,10,15,20,25};

       for (int m =0;m<6; m++)

            System.out.println("Hola! "+a[m]);

    }

 }
```

El programa no muestra errores al compilar pero si, termina con un mensaje de error cuando se lanza una excepción: **Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5**

Sin embargo, Java tiene mecanismos para excepciones que permiten ver qué excepción se ha producido e intentar recuperarse de ella.

Vamos a reescribir el programa de nuestra versión iterativa del saludo:

```
public class EjExcepciones {

    public static void main(String[] args) {

       int[] a = {5,10,15,20,25};

       try {

           for (int m =0;m<6; m++)

            System.out.println("arreglo "+a[m]);

       }

       catch(ArrayIndexOutOfBoundsException e){

           System.out.println(e.getMessage());

           System.out.println("Saludo desbordado");
```

```
        }
        finally{
            System.out.println("¡Esto se hace siempre!");
        }
    }
}
```

La palabra clave *finally* define un bloque de código que se quiere que sea ejecutado siempre, de acuerdo a si se capturó la excepción o no. En el ejemplo anterior, la salida en la consola, con `i=4` sería:

```
arreglo 5
arreglo 10
arreglo 15
arreglo 20
arreglo 25
5
Saludo desbordado
¡Esto se hace siempre!
```

GENERAR EXCEPCIONES EN JAVA

Cuando se produce un error se debería generar, o lanzar, una excepción. Para que un método en Java, pueda lanzar excepciones, hay que indicarlo expresamente.

```
void MetodoAsesino() throws NullPointerException,CaidaException
```

Se pueden definir excepciones propias, no hay por qué limitarse a las predefinidas; bastará con extender la clase **Exception** y proporcionar la funcionalidad extra que requiera el tratamiento de esa excepción.

También pueden producirse excepciones no de forma explícita como en el caso anterior, sino de forma implícita cuando se realiza alguna acción ilegal o no válida.

Las excepciones, pues, pueden originarse de dos modos: el programa hace algo ilegal (caso normal), o el programa explícitamente genera una excepción ejecutando la sentencia *throw* (caso menos normal). La sentencia throw tiene la siguiente forma:

```
    throw ObtejoExcepction;
```

El objeto ObjetoException es un objeto de una clase que extiende la clase **Exception**.

El siguiente código de ejemplo origina una excepción de división por cero:

```
11  public class Melon {
12      public static void main( String[] a ) {
13          int i=0, j=0, k;
14
15          k = i/j;     // Origina un error de division-by-zero
16          }
17      }
```

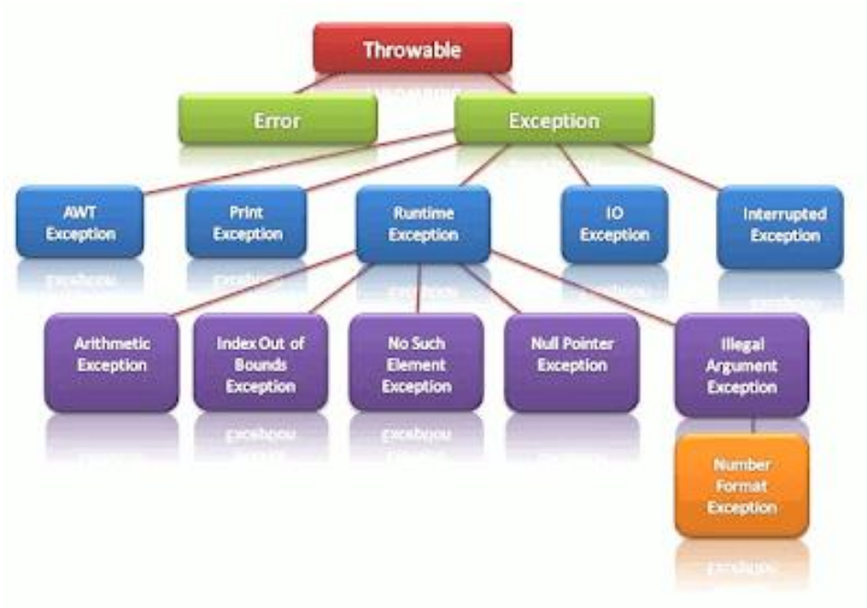
Si compilamos y ejecutamos esta aplicación Java, obtendremos la siguiente salida por pantalla:

```
Output - Atleta (run)  Melon.java
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Melon.main(Melon.java:15)
C:\Users\User\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

Las excepciones predefinidas, como *ArithmeticException*, se conocen como excepciones runtime. Actualmente, como todas las excepciones son eventos runtime, sería mejor llamarlas excepciones irrecuperables. Esto contrasta con las excepciones que generamos explícitamente, que suelen ser mucho menos severas y en la mayoría de los casos podemos recuperarnos de ellas. Por ejemplo, si un fichero no puede abrirse, preguntamos al usuario que nos indique otro fichero; o si una estructura de datos se encuentra completa, podremos sobreescribir algún elemento que ya no se necesite.

EXCEPCIONES PREDEFINIDAS

Las excepciones predefinidas y su jerarquía de clases es la que se muestra en la figura:



Los nombres de las excepciones indican la condición de error que representan. Las siguientes son las excepciones predefinidas más frecuentes que se pueden encontrar:

ArithmeticException

Las excepciones aritméticas son típicamente el resultado de una división por 0:

```
int i = 12 / 0;
```

NullPointerException

Se produce cuando se intenta acceder a una variable o método antes de ser definido:

```
class Hola extends Applet {
    Image img;

    paint( Graphics g ) {
        g.drawImage( img,25,25,this );
        }
    }
```

IncompatibleClassChangeException

El intento de cambiar una clase afectada por referencias en otros objetos, específicamente cuando esos objetos todavía no han sido recompilados.

ClassCastException

El intento de convertir un objeto a otra clase que no es válida.

```
y = (Prueba)x;       // donde
x no es de tipo Prueba
```

NegativeArraySizeException

Puede ocurrir si hay un error aritmético al intentar cambiar el tamaño de un array.

OutOfMemoryException

¡No debería producirse nunca! El intento de crear un objeto con el operador *new* ha fallado por falta de memoria. Y siempre tendría que haber memoria suficiente porque el *garbage collector* se encarga de proporcionarla al ir liberando objetos que no se usan y devolviendo memoria al sistema.

NoClassDefFoundException

Se referenció una clase que el sistema es incapaz de encontrar.

ArrayIndexOutOfBoundsException

Es la excepción que más frecuentemente se produce. Se genera al intentar acceder a un elemento de un array más allá de los límites definidos inicialmente para ese array.

UnsatisfiedLinkException

Se hizo el intento de acceder a un método nativo que no existe. Aquí no existe un método a.kk

```
class A {
    native void kk();
    }
```

y se llama a *a.kk()*, cuando debería llamar a *A.kk()*.

InternalException

Este error se reserva para eventos que no deberían ocurrir. Por definición, el usuario nunca debería ver este error y esta excepción no debería lanzarse.

CREAR EXCEPCIONES PROPIAS

También podemos lanzar nuestras propias excepciones, extendiendo la clase **System.exception**. Por ejemplo, consideremos un programa cliente/servidor. El código cliente se intenta conectar al servidor, y durante 5 segundos se espera a que conteste el servidor. Si el servidor no responde, el servidor lanzaría la excepción de time-out:

```
class ServerTimeOutException extends Exception {}

public void conectame( String nombreServidor ) throws Exception {
    int exito;
    int puerto = 80;

    exito = open( nombreServidor,puerto );
    if( exito == -1 )
        throw ServerTimeOutException;
    }
```

Si se quieren capturar las propias excepciones, se deberá utilizar la sentencia *try*:

```
public void encuentraServidor() {
   ...
   try {
        conectame( servidorDefecto );
        catch( ServerTimeOutException e ) {
            g.drawString(
                "Time-out del Servidor, intentando alternativa",
                5,5 );
            conectame( servidorAlterno );
            }
    ...
    }
```

Cualquier método que lance una excepción también debe capturarla, o declararla como parte de la interface del método. Cabe preguntarse entonces, el porqué de lanzar una excepción si hay que capturarla en el mismo método. La respuesta es que las excepciones no simplifican el trabajo del control de errores. Tienen la ventaja de que se puede tener muy localizado el control de errores y no tenemos que controlar millones de valores de retorno, pero no van más allá.

CAPTURAR EXCEPCIONES

Las excepciones lanzadas por un método que pueda hacerlo deben recoger en bloque *try/catch* o *try/finally*.

```
int valor;
try {
    for( x=0,valor = 100; x < 100; x ++ )
        valor /= x;
    }
catch( ArithmeticException e ) {
    System.out.println( "Matemáticas locas!" );
    }
catch( Exception e ) {
    System.out.println( "Se ha producido un error" );
    }
```

try

Es el bloque de código donde se prevé que se genere una excepción. Es como si dijésemos "intenta estas sentencias y mira a ver si se produce una excepción". El bloque try tiene que ir seguido, al menos, por una cláusula catch o una cláusula finally

catch

Es el código que se ejecuta cuando se produce la excepción. Es como si dijésemos "controlo cualquier excepción que coincida con mi argumento". En este bloque tendremos que asegurarnos de colocar código que no genere excepciones. Se pueden colocar sentencias catch sucesivas, cada una controlando una excepción diferente. No debería intentarse capturar todas las excepciones con una sola cláusula, como esta:

```
    catch( Excepcion e ) { ...
```

Esto representaría un uso demasiado general, podrían llegar muchas más excepciones de las esperadas. En este caso es mejor dejar que la excepción se propague hacia arriba y dar un mensaje de error al usuario.

Se pueden controlar grupos de excepciones, es decir, que se pueden controlar, a través del argumento, excepciones semejantes. Por ejemplo:

```
class Limites extends Exception {}
class demasiadoCalor extends Limites {}
class demasiadoFrio extends Limites {}
class demasiadoRapido extends Limites {}
class demasiadoCansado extends Limites {}
    .
    .
    .
    try {
        if( temp > 40 )
            throw( new demasiadoCalor() );
        if( dormir < 8 )
            throw( new demasiado Cansado() );
    } catch( Limites lim ) {
        if( lim instanceof demasiadoCalor )
            {
            System.out.println( "Capturada excesivo calor!" );
            return;
            }
        if( lim instanceof demasiadoCansado )
            {
            System.out.println( "Capturada excesivo cansancio!" );
            return;
            }
    } finally
        System.out.println( "En la clausula finally" );
```

La cláusula catch comprueba los argumentos en el mismo orden en que aparezcan en el programa. Si hay alguno que coincida, se ejecuta el bloque. El operador *instanceof* se utiliza para identificar exactamente cual ha sido la identidad de la excepción.

finally

Es el bloque de código que se ejecuta siempre, haya o no excepción. Hay una cierta controversia entre su utilidad, pero, por ejemplo, podría servir para hacer un log o un seguimiento de lo que está pasando, porque como se ejecuta siempre puede dejarnos grabado si se producen excepciones y nos hemos recuperado de ellas o no.

Este bloque finally puede ser útil cuando no hay ninguna excepción. Es un trozo de código que se ejecuta independientemente de lo que se haga en el bloque try.

Cuando vamos a tratar una excepción, se nos plantea el problema de qué acciones vamos a tomar. En la mayoría de los casos, bastará con presentar una indicación de error al usuario y un mensaje avisándolo de que se ha producido un error y que decida si quiere o no continuar con la ejecución del programa.

Por ejemplo, podríamos disponer de un diálogo como el que se presenta en el código siguiente:

```
public class DialogoError extends Dialog {
    DialogoError( Frame padre ) {
        super( padre,true );
        setLayout( new BorderLayout() );

        // Presentamos un panel con continuar o salir
        Panel p = new Panel();
        p.add( new Button( "¿Continuar?" ) );
        p.add( new Button( "Salir" ) );

        add( "Center",new Label(
            "Se ha producido un error. ¿Continuar?" ) )
        add( "South",p );
        }

    public boolean action( Event evt,Object obj ) {
        if( "Salir".equals( obj ) )
            {
            dispose();
            System.exit( 1 );
            }
        return false;
        }
    }
```

Y la invocación, desde algún lugar en que se suponga que se generarán errores, podría ser como sigue:

```
try {
    // Código peligroso
    }
catch( AlgunaExcepcion e ) {
    VentanaError = new DialogoError( this );
    VentanaError.show();
    }
```