INDICE

1. EXCEPCIONES

Los programas no están libres de errores, ni los usuarios introducen siempre los datos esperados. Por ello es conveniente poder notificar cuándo se ha producido un fallo en la ejecución de un programa y salir airosamente de la situación.

Java presenta un mecanismo conocido como manejo de excepciones (**exception handling**), que pretende que, ante un error, sea posible:

- ✓ Retornar a un estado seguro y permitir al usuario ejecutar otros comandos.
- ✓ Permitir al usuario "guardar" su trabajo y abandonar el programa de forma menos abrupta.

Una excepción es una situación anómala que puede provocar que el programa no funcione de forma correcta o termine de forma inesperada.

Los principales tipos de errores que debe controlar un programador son:

- ✓ Entrada errónea de datos por parte del usuario
- ✓ No hay memoria disponible para asignar
- ✓ Acceso a un elemento de un array fuera de rango
- ✓ Error al abrir un fichero
- ✓ División por cero
- ✓ Problemas de Hardware

Si la excepción no se trata, el manejador de excepciones del sistema realiza lo siguiente:

- ✓ Muestra la descripción de la excepción y la traza de la pila de llamadas
- ✓ Provoca el final del programa.

Los errores se pueden clasificar:

1. Errores en tiempo de compilación (compile-time errors)
   - ✓ Fallos de sintaxis al escribir los programas
   - ✓ Detectados por los compiladores
   - ✓ Deben eliminarse totalmente para poder ejecutar los programas.
2. Errores en tiempo de ejecución (run-time errors)
   - ✓ Fallos debidos a deficiencias en la creación (análisis, diseño, codificación...) de los programas
   - ✓ Deben eliminarse durante el proceso de pruebas y depuración, pero es complicado que se consiga totalmente.
   - ✓ Fallos por condiciones no controlables por los programas (E/S, desbordamientos)

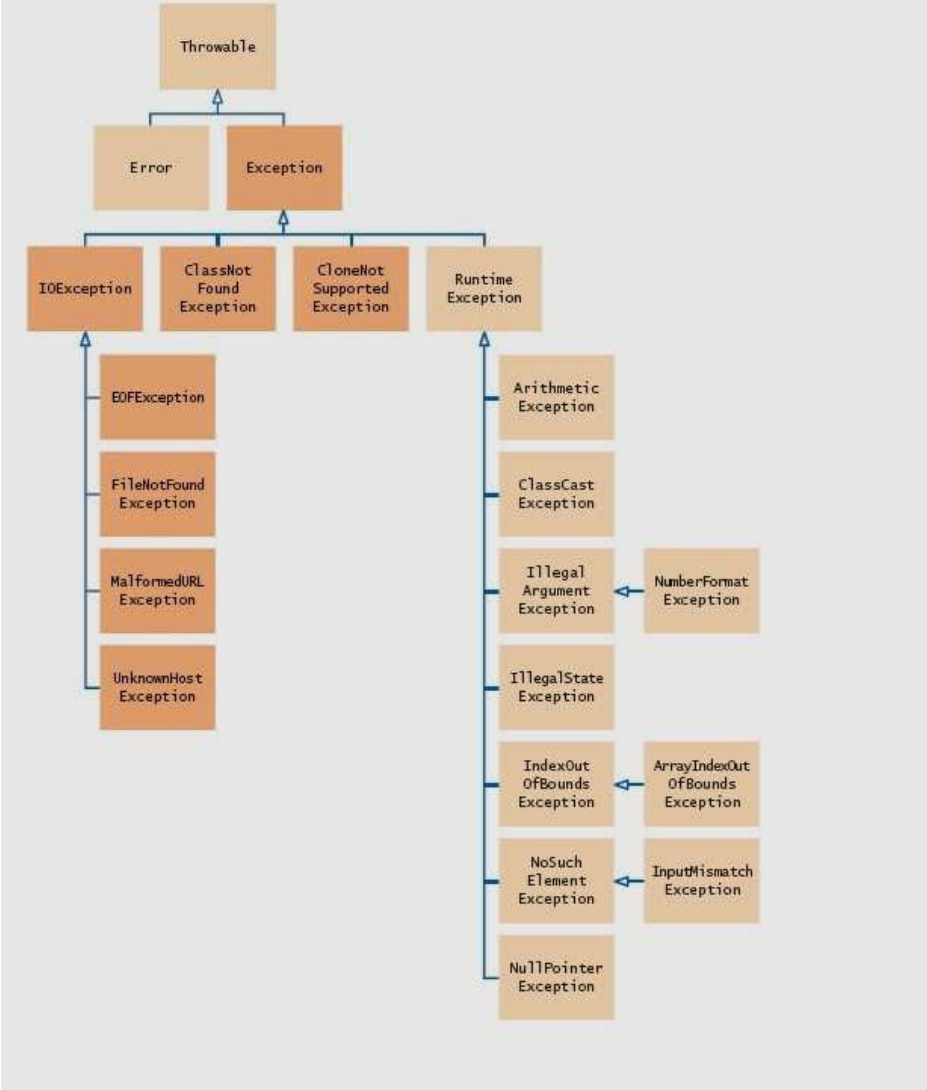
Java proporciona el manejo de excepciones para **tratar los errores en tiempo de ejecución**.

Una excepción es un **"evento"** producido en tiempo de ejecución que se lanza (**throw**) cuando se da un error.

Java proporciona mecanismos para que:

- ✓ Cualquier método pueda lanzar (throw) un objeto excepción (Exception) ante una situación anómala.
- ✓ Cualquier método pueda capturar y tratar (try…catch{ ..}) las excepciones que se produzcan en la ejecución de su código.

En Java las excepciones son objetos que encapsulan información del error que se ha producido. Tienen su propia sintaxis y forman parte de una jerarquía particular de herencia, que se presenta a continuación.



En java una excepción es siempre una instancia de una clase que hereda de la clase **Throwable**.

Java lanzará una excepción en respuesta a una situación poco usual. Cuando se produce un error se genera un objeto asociado a esa excepción. Este objeto es de la clase **Exception** o de alguna de sus herederas. Este objeto se pasa al código que se ha definido para manejar la excepción. Dicho código puede manipular las propiedades del objeto **Exception**.

Las más utilizadas son:

- ✓ **ArithmeticException**: Desbordamiento o división por cero.
- ✓ **NumberFormatException**: Conversión ilegal de String a tipo numerico
- ✓ **IndexOutOfBoundsException**: Acceso a una posición fuera de rango de una matriz o String
- ✓ **NegativeArraySizeException**: Intento de crear una matriz o vector de longitud negativa
- ✓ **NullPointException**: Intento de utilizar una referencia **null**
- ✓ **NoSuchElementException**: Intento fallido de obtener el siguiente elemento

2. CAPTURAR UNA EXCEPCIÓN

Para poder capturar excepciones, emplearemos la estructura de captura de excepciones **try-catch-finally**.

Básicamente, para capturar una excepción lo que haremos será declarar bloques de código donde es posible que ocurra una excepción. Esto lo haremos mediante un **bloque try**. Si ocurre una excepción dentro de estos bloques, se lanza una excepción. Estas excepciones lanzadas se pueden capturar por medio de **bloques catch**. Será dentro de este tipo de bloques donde se hará el manejo de las excepciones.

- ✓ El **bloque try** captura la excepción
- ✓ El **bloque catch** contiene el código que maneja la excepción

Su sintaxis es:

```
try
{
    código que puede generar excepciones;
}
catch (Tipo_excepcion1 objeto_excepcion)
{
    Manejo de excepción de Tipo_excepcion_1;
}
catch (Tipo_excepcion2 objeto_excepcion)
{
    Manejo de excepción de Tipo_excepcion_2;
}
...
finally
{
    instrucciones que se ejecutan siempre;
}
```

- ✓ El **bloque try** sólo aparece una vez.
- ✓ El **bloque catch** puede repetirse tantas veces como excepciones diferentes se deseen capturar.
- ✓ El **bloque finally** es opcional y, si aparece, solo puede aparecer una vez.
- ✓ El **código asociado a un catch** solo se ejecuta si se produce su excepción asociada

Cada **catch** maneja un tipo de excepción. Cuando se produce una excepción, se busca el **catch** que posea el manejador de excepción adecuado, será el que utilice el mismo tipo de excepción que se ha producido. Esto puede causar problemas si no se tiene cuidado, ya que la clase Exception es la

superclase de todas las demás. Por lo que si se produjo, por ejemplo, una excepción de tipo **AritmethicException** y el primer **catch** captura el tipo genérico **Exception**, será ese catch el que se ejecute y no los demás.

Por eso **el último catch** debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos. Lógicamente si vamos a tratar a todas las excepciones, sean del tipo que sean, podemos utilizar un solo catch que capture objetos **Exception**.

Parámetros de las excepciones

Las excepciones siempre se crean haciendo uso de **new**, que asigna espacio de almacenamiento e invoca a un constructor.

Hay dos constructores en todas las excepciones estándar de Java:

- ✓ Un constructor por defecto
- ✓ Un constructor que admite un parámetro de tipo **String**. Este String puede recuperarse posteriormente usando varios métodos.

Habitualmente se usará una clase de excepción para cada tipo de error.

Ejemplo:

Programa que introduce datos en un array:

```
import java.util.Scanner;
public class Ejemplo2
{
    public static void main(String argv[])
    {
        Scanner leer=new Scanner(System.in);
        int vector[] = new int[10];

        /**
         * Cargamos el array con los valores leidos por teclado. Lo hacemos dentro de un try para absorber un posible desbordamiento
         */

        try {

            for(int ind=0; ind<=vector.length; ind++)
            {
                System.out.print("Numero["+ind+"]: ");
                vector[ind]=leer.nextInt();
            }
         } catch(ArrayIndexOutOfBoundsException e)
         {
            //Esta parte del código sólo se ejecutará cuando se produzca alguna excepción de tipo ArrayIndexOutOfBoundsException
           System.out.println("Se ha intentado acceder a una posicion fuera del array");
          // aquí podemos hacer lo que queramos
         }

        /**
         * Mostramos los valores del array. Aqui podriamos incluir el codigo dentro de un try..catch
         */

        System.out.println("Los valores almacenados en el array son: ");
        for(int ind=0; ind<vector.length; ind++)
        {
            System.out.print(vector[ind]+" ");
        }
    }
}
```

Las palabras reservadas throw y throws

Hay excepciones que las lanza el propio sistema al intentar realizar una operación ilegal (como las excepciones producidas al intentar dividir por 0) y otras las debe lanzar el programador, siendo este el encargado de crear y generar excepciones definidas por él mismo.

La mayor parte de las excepciones pertenecen a la categoría de **excepciones comprobadas estándar**. Si un bloque de código puede generar una excepción de este tipo, entonces el programador debe proporcionar un bloque catch para ella, o indicar que bebe propagar la excepción, incluyendo una clausula **throws** en la declaración del método.

Las principales excepciones capturadas estándar son:

- ✓ **java.io.IOException:** incluye la mayoria de las excepciones de entrada y salida.
- ✓ **java.io.EOFException:** Encontrado fin de archivo antes de completar la entrada
- ✓ **java.io.FileNotFoundException:** No se ha encontrado el archivo para abrirlo

La palabra reservada **throws** se utiliza para indicar al compilador que una función puede disparar una excepción manualmente.

La palabra reservada **thow** se usa para disparar una excepción manualmente.

3. CAPTURAR UNA EXCEPCIÓN CON THROWS

Hemos visto cómo capturar excepciones que se produzcan en el código, pero en lugar de capturarlas también podemos hacer que se propaguen al método de nivel superior, desde el cual se ha llamado al método actual. Para esto, en el método donde se vaya a lanzar la excepción, se siguen 3 pasos:

1. Indicar en el método que determinados tipos de excepciones o grupos de ellas pueden ser lanzados, cosa que haremos usando la palabra reservada **throws**, de la siguiente forma, por ejemplo:

```
public void lee_fichero()
   throws IOException, FileNotFoundException
{
   // Cuerpo de la función
}
```

   Podremos indicar tantos tipos de excepciones como queramos en la clausula **throws**. Si alguna de estas clases de excepciones tiene subclases, también se considerará que puede lanzar todas estas subclases.

2. Para lanzar la excepción utilizamos la instrucción **throw**, proporcionándole un objeto correspondiente al tipo de excepción que deseamos lanzar. Por ejemplo:

```
public void lee_fichero()
   throws IOException, FileNotFoundException
{
   ...
   throw new IOException(mensaje_error);
   ...
}
```

   Podremos lanzar así excepciones en nuestras funciones para indicar que algo no es como debiera ser a las funciones llamadoras. El método en el que se propaga la excepción pasará la

excepción al método de nivel superior y así hasta llegar al programa principal, momento en el que, si no se captura la excepción, provocará la salida del programa.

3. En la función principal se debe colocar el catch que procesa la instrucción:

```
public static void main()
{
    ......
    try
    {
        lee_fichero();
    }catch(IOException e)
    {
        //Tratamiento de la excepción de entrada y salida
    }catch(FileNotFoundException ex)
    {
        //Tratamiento de la excepción de fichero no encontrado
    }
    .......
}
```

Esto hace que el tratamiento de excepciones esté centralizado y produce código más limpio.

Ejemplo:

```
//Metodo que devuelve la cantidad de divisores de un numero que hay almacenados en un array. Si se intenta dividir por cero
//el metodo no trata la excepción pasa el tratamiento al main
public static int esDivisor(int num, int vector[]) throws java.lang.ArithmeticException
{
    int acum=0;

    for(int ind=0; ind<vector.length; ind++)
    {
        if((num%vector[ind])==0)
        {
            acum++;
        }
    }

    return acum;
}
```

```
public static void main(String args[])
{
    int num=648, divisores=0;
    int vector[]={57,18,0,34,26,83,45,0,24,33};

    //Llamamos a la función dentro del try
    try
    {
       divisores=esDivisor(num,vector);
    }
    catch(java.lang.ArithmeticException e)
    {
        System.out.println("Se ha intentado dividir por cero");
    }

    System.out.println("Hay almacenados "+divisores+" numeros divisores de "+num);

}
```

En general, hay dos formas de manejar las excepciones:

- ✓ **Interrupción.** En este caso se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y se entiende que no hay manera de regresar al código que provocó la excepción.
- ✓ **Reanudación.** Se maneja el error y se regresa de nuevo al código que provocó la excepción.

La filosofía de Java es del tipo interrupción, pero se puede simular la reanudación por medio de bucles.

Ejemplo:

Programa que pide números por teclado y muestra un mensaje indicando si el número 648 es divisible por el numero leído. El programa seguirá ejecutándose mientras indiquemos que queremos seguir. Se captura y se trata la excepción de intentar dividir por 0

La siguiente implementación saldrá del programa al encontrar la primera excepción:

```
Scanner leer=new Scanner(System.in);
int num;
char seguir='s';

try
{
    do
    {
        System.out.print("Dame un numero: ");
        num=leer.nextInt();

        if((684%num)==0)
        {
            System.out.println("El numero "+num+" es divisor de 648");
        }
        else
        {
            System.out.println("El numero "+num+" no es divisor de 648");
        }

        System.out.print("Pulsa 's' para seguir, cualquier otro caracter para terminar");
        seguir=leer.next().charAt(0);

    }while(seguir=='s');
}
catch(java.lang.ArithmeticException e)
{
    //Si hemos introducido un 0 el progrma nos indicará que hemos intentado dividir por 0 y el programa terminará de forma controlada

    System.out.println("La excepcion generada ha sido "+e.getMessage());
}
finally
{
    //El finally se ejecutará cuando se produzca cualquier error: porque hayamos introducido un 0 como divisor o porque hayamos introducido
    //un valor numerico en seguir
    System.out.println("Programa finalizado");
}
```

Si introducimos el **try-catch** dentro del bucle, después de tratar la excepción se seguirá ejecutando el código que provocó la excepción. El código anterior quedaría así.

```
Scanner leer=new Scanner(System.in);
int num;    //para almacenar el numero leido por teclado y comprobar si es un divisor de 648
char seguir='s';
do{
    try
    {
        //Si el numero introducido es un valor numérico se comprueba si es divisor de 648. Si es un valor alfabetico se termina el programa
        System.out.print("Dame un numero (para terminar introduce un caracter): ");
        num=leer.nextInt();
        leer.nextLine();   //limpiamos el buffer

        if((684%num)==0)
        {
            System.out.println("El numero "+num+" es divisor de 648");
        }
        else
        {
            System.out.println("El numero "+num+" no es divisor de 648");
        }
```

```
    }
    catch(java.lang.ArithmeticException e)
    {
        //Este mensaje aparecerá cuando introduzcamos un numero 0. Nos mostrará el mensaje por defecto de la clase
        // y se seguirá ejecutando el programa
        System.out.println(e.getMessage());
    }
    catch(java.util.InputMismatchException ex)
    {
        //Si hemos introducido un valor de tipo caracter en vez de numerico indicamos que quereos terminar
        //En seguir se introduce una 'n' para terminar el bucle
        seguir='n';
    }

}while(seguir=='s');
```

En estos casos, tras el tratamiento de la excepción, no se reanuda la ejecución del código que provocó la excepción.