

Colecciones

Dr. Mario Marcelo Berón

Índice

1 Archivos

2 Excepciones

Concepto



- Una forma de organizar la información de manera tal que perdure en el tiempo.
- A bajo nivel se puede decir que es: *Un conjunto de bits almacenados en un dispositivo, y accesible a través de un camino de acceso (path) que lo identifica.*

Concepto



- Si bien en definitiva todos los archivos almacenan 0s y 1s (bytes), no es lo mismo acceder a bits que a strings, es por esto que se puede distinguir dos tipos de archivos:
 - ▶ Archivos de Texto.
 - ▶ Archivos Binarios.

Archivos



La forma de interactuar con los sistemas de archivos locales se realiza a través de la clase *File*, esta clase proporciona muchas utilidades relacionadas con archivos y con la obtención de información básica sobre esos archivos.

Enfoque Clásico: Devolución de un Código de Diagnóstico



//Declaración

```
File miArchivo;
```

//Una forma

```
miArchivo = new File( "path/mi_archivo" );
```

//Otra forma

```
miArchivo = new File( "path", "mi_archivo" );
```

Comprobaciones y Utilidades

- Relacionadas con el nombre del archivo:
 - ▶ `String getName()`
 - ▶ `String getPath()`
 - ▶ `String getAbsolutePath()`
 - ▶ `String getParent()`
 - ▶ `boolean renameTo(File nuevoNombre)`
- Comprobaciones:
 - ▶ `boolean exists()`
 - ▶ `boolean canWrite()`
 - ▶ `boolean canRead()`
 - ▶ `boolean isFile()`
 - ▶ `boolean isDirectory()`
 - ▶ `boolean isAbsolute()`

Comprobaciones y Utilidades



- Información General del Archivo:
 - ▶ `long lastModified()`
 - ▶ `long length()`
- Utilidades de Directorio:
 - ▶ `boolean mkdir()`
 - ▶ `String[] list()`

Ejemplo

```
public static void main( String args[] )
    throws IOException {
    if( args.length > 0 ){
        for( int i=0; i < args.length; i++ ){
            File f = new File( args[i] );
            if( f.exists() ){
                System.out.println( "Nombre: "+f.getName() );
                System.out.println( "Camino: "+f.getPath() );
                System.out.print(" Leer:"+f.canRead());
                System.out.print(" Escribir:"+f.canWrite());
                System.out.println( "." );
                System.out.println( "Longitud:"+f.length()+" bytes"
                    );
            }
        }
        ...
    }
```

Apertura de un Archivo de Texto



```
File f= new File ( "miarchivo_texto.txt");  
  if (f.exists()){  
    //trabajo con el archivo  
  }
```

Lectura de un Archivo de Texto



```
File f= new File ("Leer.txt");  
FileReader leo= new FileReader(f);  
BufferedReader bLeo= new BufferedReader(leo);  
String s;  
  while((s=bLeo.readLine())!=null){  
    texto += s + "\n ";  
  }  
System.out.println(texto)
```

Escritura de un Archivo de Texto



```
File escribir= new File ("Escribir.txt");  
FileWriter fwescribir= new FileWriter(escribir);  
BufferedWriter bwescribir= new BufferedWriter (  
    fwescribir);  
    bwescribir.append("hola que tal\n");
```

Importante

Si se desea agregar texto al final del archivo se debe crear un `FileWriter` de la siguiente manera:

```
FileWriter(escribir, true);
```

Excepciones - Conceptos



- Programación a la defensiva
 - ▶ Anticiparse a lo que podría ir mal.
- Lanzamiento y tratamiento de excepciones.

Causas de Situaciones de Error



- Implementación incorrecta
 - ▶ No se ajusta a las especificaciones.
- Requerimiento de objeto inapropiada.
 - ▶ Índice Inválido.
 - ▶ Referencia nula.
- Estado de un objeto inapropiado o inconsistente.

No Siempre son Errores de Programación



- Errores del Entorno
 - ▶ URL incorrecta.
 - ▶ Interrupción de las comunicaciones de red.
- Requerimiento de objeto inapropiada.
 - ▶ Archivos que no existen.
 - ▶ Permisos inapropiados.

¿Cómo informar los errores?

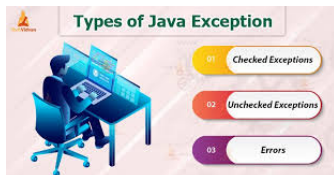


- Al usuario
 - ▶ ¿Se puede resolver el problema?.
- Al objeto
 - ▶ Devuelve un valor de diagnóstico
 - ▶ Dispara una excepción

Enfoque Clásico: Devolución de un Código de Diagnóstico

```
public double calcularPorcentaje(double valor[],
                                int li, int ls,
                                double porcentaje ){
    if (li >= 0 && ls < valor.length)
    {
        int i;
        double r;
        r=0;
        for (i=li; i<=ls; i++)
            r=r+valor[i];
        return r*porcentaje;
    }
    else return -1;
}
```

Problemas con el Enfoque Clásico



- Mecanismo simple
- Complica el código (cuando se trabaja con varias funciones los if-else se comienzan a anidar)
- Hay errores de ejecución que no pueden capturarse fácilmente
 - ▶ Por ejemplo, en C++ al quedarse sin memoria al hacer new o cuando se cae una conexión

Excepciones



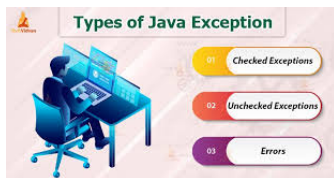
- ❶ En lenguajes de POO modernos se usan excepciones.
- ❷ El código se estructura:
 - ▶ En el código del objeto se puede indicar la ocurrencia de una excepción y esto ocasiona la finalización inmediata del método:
 - ❶ Por ejemplo el incumplimiento de alguna condición necesaria para la ejecución del método
 - ❷ También pueden ocurrir excepciones del entorno (no originadas explícitamente por la aplicación): i) Síncronas: división por cero, acceso fuera de los límites de un array, puntero nulo; ii) Asíncronas: algún fallo del sistema, caída de comunicaciones, falta de memoria.

Excepciones



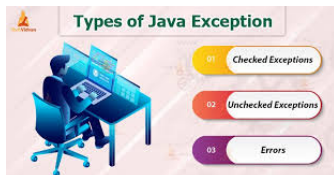
- En lenguajes de POO modernos se usan excepciones.
- El código se estructura:
 - ▶ El código del objeto se puede estructurar en dos partes:
 - ★ Secuencia normal de ejecuciones.
 - ★ Tratamiento de Excepciones: ¿qué hacer cuando se sale del flujo normal de ejecución por la ocurrencia de alguna excepción?

Excepciones en Java



- Las excepciones son objetos de la clase `Exception` que hereda de `Throwable`
- En Java puede haber dos tipos de excepciones:
 - ▶ Excepciones que no requieren comprobarse
 - ▶ Excepciones que hay que comprobar
- Cuando ocurre una excepción se dice que se lanza (throw) `throw new Excepción();`
- La excepción puede ser capturada para tratarla (catch) `catch (Excepción e) { tratamiento(); }`

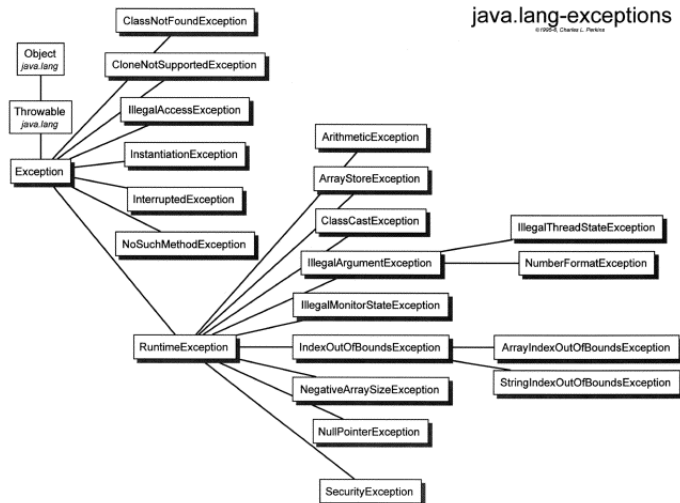
Excepciones en Java



Sin comprobación obligatoria:

- Subclases de RuntimeException
- Son las que puede lanzar la MVJ
 - ▶ Normalmente representan una condición fatal del programa
 - ▶ No las comprueba el compilador y es difícil saber cuándo y por qué pueden suceder.
 - ▶ Pero a veces se puede considerar que alguna condición podría ocasionarlas.

Excepciones en Java



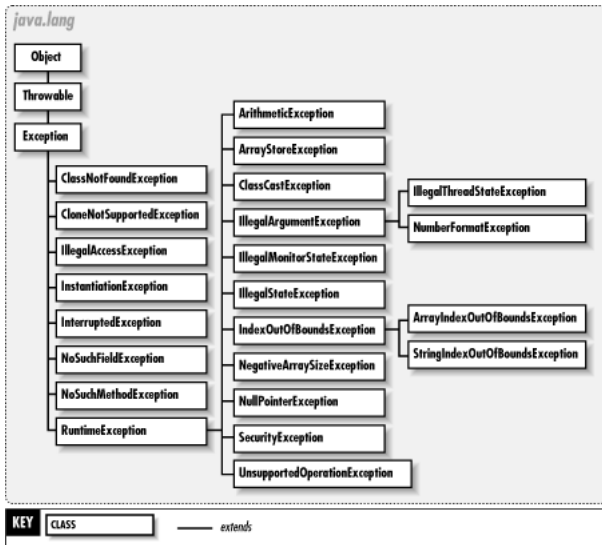
Categorías de Excepción



Con comprobación:

- Subclases de Exception
- Las comprueba el compilador
 - ▶ Si no se consideran en el código, el compilador indica un error
 - ▶ Estas excepciones son lanzadas por métodos que se usan en el código.

Excepciones en Java

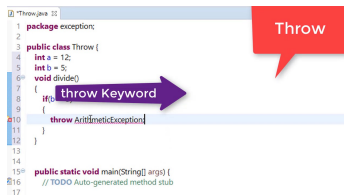


Lanzamiento de una Excepción

```
public double calcularPorcentaje(double valor[],
                                int li, int ls,
                                double porcentaje )
    ..{

if (li >= 0 && ls < valor.length)
{
    int i;
    double r;
    r=0;
    for (i=li; i<ls; i++)
        r=r+valor[i];
    return r*porcentaje;
}
else
    throw new ArrayIndexOutOfBoundsException();
}
```

Lanzamiento de una Excepción



The screenshot shows a Java code editor with the following code:

```
1 package exception;
2
3 public class Throw {
4     int a = 12;
5     int b = 5;
6     void divide()
7     {
8         // throw Keyword
9         {
10             throw ArithmeticException;
11         }
12     }
13
14
15     public static void main(String[] args) {
16         // TODO Auto-generated method stub
17     }
18 }
```

A red speech bubble labeled "Throw" points to the `Throw` class name. A purple arrow labeled "throw Keyword" points to the `throw` keyword in the `divide()` method.

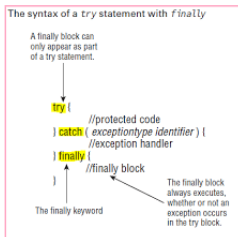
- El método lanzador finaliza prematuramente.
- No devuelve ningún valor de retorno.
- El control no vuelve al punto de llamada.
 - ▶ El lanzador no puede despreocuparse.
 - ▶ Se puede capturar (catch) la excepción

Lanzamiento de una Excepción

Los métodos que lanzan una excepción controlada deben declararla con la clausula *throw*.

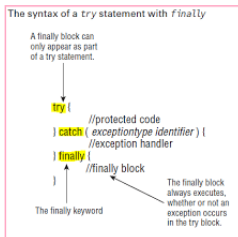
```
public void saveToFile(String destinationFile)
    throws IOException {...}
public double cPorcentaje(double valor[],
    int li, int ls, double porcentaje)
    throws ArrayIndexOutOfBoundsException{
    if (li >= 0 && ls < valor.length){
        int i;        double r;
        r=0;
        for (i=li; i<ls; i++) r=r+valor[i];
        return r*porcentaje;
    }
    else    throw new ArrayIndexOutOfBoundsException()
        ;
}
```

Tratamiento de Excepciones



- Tratarla en el método que las captura.
- Propagar al método llamante.
 - ▶ Si al final nadie captura la excepción, el programa finaliza y se lista la traza de la pila de llamadas.

Tratamiento de Excepciones



Sino se propagan se manejan con un bloque try catch

```
public void unMetodo() {  
    try {  
        //Aquí se puede lanzar la excepción e  
    } catch (Exception e) {  
        //código que gestiona la excepción e  
    }  
}
```

Tratamiento de Excepciones

```
public class ClaseExcepciones {  
  
    public static void main(String[] args) {  
        /*Clase donde está definida calcularPorcentaje*/  
        Excepcion e= new Excepcion();  
        double []v= new double [5];  
        try {  
            /*Cargar el arreglo v con valores*/  
            ...  
            System.out.println(e.calcularPorcentaje(v, -1, 5, 1));  
        } catch (ArrayIndexOutOfBoundsException ex) {  
            System.out.println("He capturado la excepcion");  
        }  
    }  
}
```

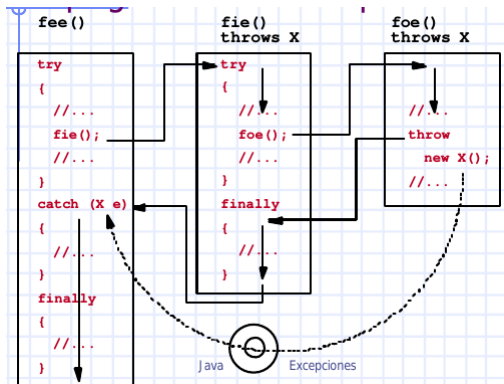
Captura de Excepciones

```
public void unMétodo(){  
    try{  
        // se puede lanzar la excepción e1 o e2  
    } catch(MiExcepción e1){  
        // código que gestiona la exception e1  
    } catch(Exception e2){  
        // código que gestiona la exception e2  
    }  
}
```

Clausula finally

Se ejecuta al final después del último catch generalmente para hacer limpieza.

Propagación de Excepciones



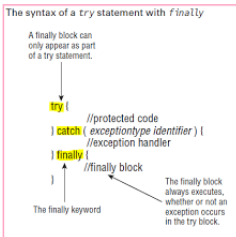
Gestión Genérica de Excepciones

- Se puede capturar una excepción genérica para responder a excepciones que sean de ese tipo de subtipos.

Ejemplos

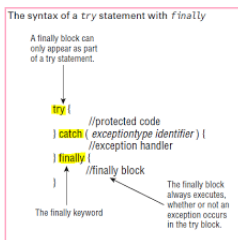
- ▶ Con la captura de `Exception` se puede responder a todas las excepciones controladas.
- ▶ Con la captura de `Throwable` se puede responder a todas las excepciones y errores.

Gestión Genérica de Excepciones



```
public void miMétodo() {  
    try {  
        //código  
    } catch (Throwable e) {  
        System.out.println(e.printStackTrace());  
    }  
}
```

Excepciones Definidas por el Usuario



- Se puede extender RuntimeException para excepciones no comprobadas.
- Se puede extender Exception para excepciones controladas.

Excepciones Definidas por el Usuario

```
public class EdadNoValida extends RuntimeException
{
    private String mensaje;
    public EdadNoValida(String m)
    { this.mensaje = m; }

    public String getMensaje()
    {return mensaje; }

    public String toString()
    {
        return "Error:" + mensaje;
    }
}
```

Excepciones Definidas por el Usuario

```
public class ClaseExcepciones {  
    public static void main(String[] args)  
        throws EdadNoValida {  
  
        int edad;  
        /* Ingresar valores a la una variable edad*/  
        if (edad >200)  
            throw new EdadNoValida("Un humano no puede  
                                    vivir mas de 200 anios");  
        }  
        .....  
    }  
}
```

Excepciones Definidas por el Usuario

```
public class ClaseExcepciones {  
    .....  
}
```

Salida

```
Exception in thread main Error:Un humano no  
puede vivir mas de 200 años at claseexcep-  
ciones.ClaseExcepciones.main(ClaseExcepciones.java:36)  
/home/mberon/.cache/netbeans/8.1/executor-snippets/run.xml:53:  
Java returned: 1 BUILD FAILED (total time: 0 seconds)
```

Excepciones Definidas por el Usuario

```
public static void main(String[] args)
    throws EdadNoValida {
    .....
    int edad;
    try{
        /* Ingresar valor a la variable edad */
        if (edad >200)
            throw
                new EdadNoValida("Un humano no puede vivir mas
                                   de 200 anios");

        ....
    }catch (EdadNoValida env){
        System.out.println("Aqui se controla la
                            excepcion EdadNoValida");
    }
}
```


Excepciones Definidas por el Usuario

```
public static void main(String[] args)
    throws EdadNoValida {
    .....
}
```

Salida

Aqui se controla la excepcion EdadNoValida BUILD SUCCESSFUL
(total time: 0 seconds)

Ventajas del Mecanismo de Excepciones

- Separación del tratamiento de errores del resto del código del programa
 - ▶ Flujo del programa más sencillo
 - ▶ Evita manejos de códigos de error
- Propagación de errores a lo largo de la pila de llamadas a métodos
 - ▶ Evitar retornos continuos en caso de error
 - ▶ Evitar la necesidad de argumentos adicionales
- Agrupación y definición de tipos de errores como clases
 - ▶ Jerarquía de clases
 - ▶ Tratar errores a diferentes niveles de especificidad