http://www.arrakis.es/~abelp/ApuntesJava/GestionExcepciones.htm

Gestión de Excepciones

Excepciones. Categorías.

Las excepciones son el mecanismo por el cual pueden controlarse en un programa Java las condiciones de error que se producen. Estas condiciones de error pueden ser errores en la lógica del programa como un índice de un array fuera de su rango, una división por cero o errores disparados por los propios objetos que denuncian algún tipo de estado no previsto, o condición que no pueden manejar.

La idea general es que cuando un objeto encuentra una condición que no sabe manejar crea y dispara una excepción que deberá ser capturada por el que le llamó o por alguien más arriba en la pila de llamadas. Las excepciones son objetos que contienen información del error que se ha producido y que heredan de la clase Throwable o de la clase Exception. Si nadie captura la excepción interviene un manejador por defecto que normalmente imprime información que ayuda a encontrar quién produjo la excepción.

Existen dos categorías de excepciones:

- Excepciones verificadas: El compilador obliga a verificarlas. Son todas las que son lanzadas explicitamente por objetos de usuario.
- Excepciones no verificadas: El compilador no obliga a su verificación. Son excepciones como divisiones por cero, excepciones de puntero nulo, o índices fuera de rango.

Generación de excepciones

Supongamos que tenemos una clase Empresa que tiene un array de objetos Empleado (clase vista en capítulos anteriores). En esta clase podríamos tener métodos para contratar un Empleado (añadir un nuevo objeto al array), despedirlo (quilarlo del array) u obtener el nombre a partir del número de empleado. La clase podría ser algo así como lo siguiente:

```
public class Empresa {
    String nombre;
    Empleado [] listaEmpleados;
    int totalEmpleados = 0;
    . . .
    Empresa(String n, int maxEmp) {
        nombre = n;
        listaEmpleados = new Empleado [maxEmp];
    }
    . . .
    void nuevoEmpleado(String nombre, int sueldo) {
       if (totalEmpleados < listaEmpleados.length ) {
            listaEmpleados[totalEmpleados++]
```

```
= new Empleado(nombre,sueldo);
        }
    }
}
```

Observese en el método nuevoEmpleado que se comprueba que hay sitio en el array para almacenar la referencia al nuevo empleado. Si lo hay se crea el objeto. Pero si no lo hay el método no hace nada más. No da ninguna indicación de si la operación ha tenido éxito o no. Se podría hacer una modificación para que, por ejemplo el método devolviera un valor booleano true si la operación se ha completado con éxito y false si ha habido algún problema.

Otra posibilidad es generar una excepción verificada (Una excepción no verificada se produciría si no se comprobara si el nuevo empleado va a caber o no en el array). Vamos a ver como se haría esto.

Las excepciones son clases, que heredan de la clase genérica Exception. Es necesario por tanto asignar un nombre a nuestra excepción. Se suelen asignar nombres que den alguna idea del tipo de error que controlan. En nuestro ejemplo le vamos a llamar CapacidadEmpresaExcedida.

Para que un método lance una excepción:

- Debe declarar el tipo de excepción que lanza con la cláusula throws, en su declaración.
- Debe lanzar la excepción, en el punto del código adecuado con la sentencia throw.

En nuestro ejemplo:

```
    void nuevoEmpleado(String
nombre, int sueldo) throws CapacidadEmpresaExcedida{
        if (totalEmpleados < listaEmpleados.length) {
            listaEmpleados[totalEmpleados++]
= new Empleado(nombre,sueldo);
        }
        else throw new CapacidadEmpresaExcedida(nombre);
    }
```

Además, necesitamos escribir la clase `CapacidadEmpresaExcedida.` Sería algo así:

```
public class CapacidadEmpresaExcedida extends Exception {
    CapacidadEmpresaExcedida(String nombre) {
        super("No es posible añadir el empleado " +
nombre);
    }
    . . .
}
```

La sentencia throw crea un objeto de la clase `CapacidadEmpresaExcedida` . El constructor tiene un argumento (el nombre del empleado). El constructor simplemente llama al constructor de la superclase pasándole como argumento un texto explicativo del error (y el nombre del empleado que no se ha podido añadir).

La clase de la excepción puede declarar otros métodos o guardar datos de depuración que se consideren oportunos. El único requisito es que extienda la clase Exception. Consultar la documentación del API para ver una descripción completa de la clase Exception.

De esta forma se pueden construir métodos que generen excepciones.

Captura de excepciones

Con la primera versión del método nuevoEmpleado (sin excepción) se invocaría este método de la siguiente forma:

```
Empresa em = new Empresa("La Mundial");
em.nuevoEmpleado("Adán Primero",500);
```

Si se utilizara este formato en el segundo caso (con excepción) el compilador produciría un error indicando que no se ha capturado la excepción verificada lanzada por el método nuevoEmpleado. Para capturar la excepción es utiliza la construcción try / catch, de la siguiente forma:

```
Empresa em = new Empresa("La Mundial");
try {
    em.nuevoEmpleado("Adán Primero",500);
} catch (CapacidadEmpresaExcedida exc) {
    System.out.println(exc.toString());
    System.exit(1);
}
```

- Se encierra el código que puede lanzar la excepción en un bloque try / catch.
- A continuación del catch se indica que tipo de excepción se va a capturar.
- Después del catch se escribe el código que se ejecutará si se lanza la excepción.
- Si no se lanza la excepción el bloque catch no se ejecuta.

El formato general del bloque try / catch es:

```
try {
    . . .
} catch (Clase_Excepcion nombre) { . . .}
  catch (Clase_Excepcion nombre) { . . .}
   . . .
```

Observese que se puede capturar más de un tipo de excepción declarando más de una sentencia catch. También se puede capturar una excepción genérica (clase Exception) que engloba a todas las demás.

En ocasiones el código que llama a un método que dispara una excepción tampoco puede (o sabe) manejar esa excepción. Si no sabe que hacer con ella puede de nuevo lanzarla hacia arriba en la pila de llamada para que la gestione quien le llamo (que a su vez puede capturarla o reenviarla). Cuando un método no tiene intención de capturar la excepción debe declararla metdiante la cláusula throws, tal como hemos visto en el método que genera la excepción.

Supongamos que, en nuestro ejemplo es el método main de una clase el que invoca el método nuevoEmpleado. Si no quiere capturar la excepción debe hacer lo siguiente:

```
public static void main(String []
args) throws CapacidadEmpresaExcedida {
    Empresa em = new Empresa("La Mundial");
    em.nuevoEmpleado("Adán Primero",500);
}
```

Cláusula `finally`

La cláusula `finally` forma parte del bloque **`try`** / **`catch`** y sirve para especificar un bloque de código que se ejecutará tanto si se lanza la excepción como si no. Puede servir para limpieza del estado interno de los objetos afectados o para liberar recursos externos (descriptores de fichero, por ejemplo). La sintaxis global del bloque **`try`** / **`catch`** / **`finally`** es:

```
try {
    . . .
} catch (Clase_Excepcion nombre) { . . .}
  catch (Clase_Excepcion nombre) { . . .}
   . . .
  finally { . . .}
```