Excepciones

En programación, una excepción se refiere a un evento o condición anormal que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de ejecución. Puede ser causada por diversos factores, como errores de programación, entradas de datos incorrectas o condiciones imprevistas en el entorno de ejecución.

Cuando se produce una excepción, el programa normalmente detiene su ejecución en el punto donde ocurrió la excepción y busca un mecanismo para manejarla. El manejo de excepciones permite controlar y responder de manera adecuada a situaciones excepcionales, en lugar de simplemente detenerse y mostrar un error al usuario.

En resumen, hay que entender que las excepciones son para que las maneje el programador, el usuario no deberia de saltarle ningun error, ni que se le detenga el programa ni nada eso. Estas son normales, pero cada vez que hacemos algo, hay que tener en cuenta las excepciones que pueden ocurrir para hacer una cosa u otra.

Excepciones hay muchisimas y es imposible saberselas todas. Lo mejor es que cuando planeemos el codigo que vamos a escribir revisar los metodos que gastamos y las excepciones que pueden lanzar, por ejemplo si con el *Scanner* usamos *nextInt()* y el usuario introduce una cadena nos lanzara una excepcion de tipo *InputMismatchException*. Si realizamos ciertas operaciones, division, acceso a un vector... hay que tener en cuenta las que nos pueden saltar y lo mejor es probando diferentes casos.

TRUCO: cuando utilicemos un metodo, si ponemos el raton encima y lo dejamos parado nos sale la documentacion de ese metodo, ahi estan las excepciones que puede lanzar.

Manejo de excepciones

En Java el manejo de excepciones se hace mediante try/catch/finally, que tiene la siguiente sintaxis:

```
try {
    // codigo que ejecutamos

} catch (/* NombreExcepcion variable */) {
    // codigo que se ejecuta si ocurre la excepcion Exception en el bloque try

} finally {
    // codigo que se ejecuta al final, ocurra una excepcion o no
}
```

Por ejemplo, forzamos la division por cero:

```
try {
    int a = 1 / 0;

} catch (ArithmeticException e) {
    System.out.println("No se puede dividir por cero");
}
// El finally es un bloque opcional
```

Jerarquia de excepciones

Esto es importante por lo que se vera despues. En Java TODO son objetos, las excepciones tambien. La excepcion *madre* es ***Exception*** de la que heredan las demas excepciones. Por ejemplo si estamos manejando un fichero y durante la ejecucion de nuestro codigo pueden ocurrir las excepciones siguientes:

- ***IOException***: Error de entrada salida
- ***EOFException***: End-of-file
- ***FileNotFoundException***: No existe el archivo

Las dos ultimas excepciones heredan de *IOException* por que son errores de entrada/salida y lo mas importante por que Java asi lo ha decidido.

Orden de manejo de excepciones

Cuando el programa se esta ejecutando dentro de un try-catch y ocurre una excepcion hay un orden en el que el programa dice vale, hay una excepcion voy a ver que tengo que hacer, y es el siguiente:

```
try {
    // codigo que ejecutamos

} catch (/* NombreExcepcion variable */) {
    // 1 en prioridad

} catch (/* NombreExcepcion variable */) {
    // 2 en prioridad

} catch (/* NombreExcepcion variable */) {
    // 3 en prioridad

} catch (/* NombreExcepcion variable */) {
    // 4 en prioridad

}
// y todas las que queramos
```

Por tanto, lo siguiente seria un codigo mal escrito:

```
try {
    // codigo que ejecutamos, manejando ficheros

} catch (IOException e) {
    // manejo de la excepcion

} catch (FileNotFoundException e) {
    // manejo de la excepcion

} catch (EOFException e) {
    // manejo de la excepcion
}
```

Estaria mal escrito por que ***FileNotFoundException*** y ***EOFException*** heredan de ***IOException*** entonces al saltar la excepcion ***FileNotFoundException***, por ejemplo, la ejecucion se iria al bloque catch de ***IOException***. Hay que tener cuidado con esto. El codigo corregido seria asi:

```
try {
    // codigo que ejecutamos, manejando ficheros

} catch (FileNotFoundException e) {
    // manejo de la excepcion

} catch (EOFException e) {
    // manejo de la excepcion

} catch (IOException e) {
```

```
    // manejo de la excepcion
}
```

FileNotFoundException y ***EOFException*** aunque hereden de ***IOException*** son diferentes y da igual que pongamos una encima de la otra.

Lanzar excepciones

Es interesante saber que nosotros como programadores podemos lanzar excepciones tambien. Antes he dicho que las excepciones son para el manejo del programador y puede parecer una idea tonta que nosotros lancemos excepciones, pero no lo es.

La sintaxis para lanzar una excepcion es:

```
throws new Exception();
```

Y en el metodo que la lanza hay que indicarlo:

```
public int division(int a, int b) throws ArithmeticException {
    // Codigo del metodo
}
```

Por ejemplo, si nosotros tenemos una clase Coche con los atributos ***modelo*** y ***numeroRuedas***. Ahora bien, queremos limitar el numero de ruedas a 3 o 4, ya que si tuviera 2, seria una moto y 1 o mas de 4 imposible. Esto se hace añadiendo un filtro en el ***setter*** de ***numeroRuedas***, en que nos lance una excepcion ***IllegalArgumentException***, que es una excepcion predefinida de Java que se utiliza para esto. La clase quedaria asi:

```
public class Coche {
    String modelo;
    int numeroRuedas;

    // Aqui habria que poner el throws tambien por que no la estamos manejando,
    //  ni tenemos que hacerlo
    public Coche(String modelo, int numeroRuedas) throws IllegalArgumentException{
        this.modelo = modelo;
        setNumeroRuedas(numeroRuedas);      // esto es lo que lanza la excepcion
    }

    public void setNumeroRuedas(int numeroRuedas) throws IllegalArgumentException {
        // si es diferente de 3 y 4 se lanza la excepcion
        if (numeroRuedas != 3 && numeroRuedas != 4)
            throw new IllegalArgumentException("El numero de ruedas debe ser 3 o 4");

        // si la excepcion salta aqui no llega
        this.numeroRuedas = numeroRuedas;
    }
}
```

NOTA: podriamos lanzar cualquier excepcion pero para la legibilidad del codigo es interesante que lancemos excepciones con nombres que por lo menos se asemejen al problema real

Y en el programa principal o en otro metodo en el que instanciemos un objeto de la clase coche el manejo seria asi:

```
try {
    String modelo = leeModelo();     // lo pide al usuario
    int ruedas = leeRuedas();        // lo pide al usuario
```

```
    Coche c = new Coche(modelo, ruedas);

    // do something

} catch (IllegalArgumentException e) {
    System.out.println("Error al instanciar el coche: " + e.getMessage());
}
```

Si os fijais gasto ***e.getMessage()***, esto es interesante por que imaginad que tambien limito los valores de el atributo *modelo*, entonces en su setter, al hacer el throw le meteria otro mensaje y dependiendo de cual ocurra imprimiria un mensaje u otro para avisar al usuario de lo que ha hecho mal.

Excepciones personalizadas

Para crear excepciones personalizadas simplemente tenemos que crear una clase que herede de ***Exception*** o de cualquier otra excepcion. Hay que tener en cuenta el tema del manejo de la herencia que se explico anteriormente si decidimos que herede de otra.

```
class RuedasInvalidasException extends Exception {
    public RuedasInvalidasException(String msg) {
        super(msg);
    }
}

class ModeloInvalidoException extends Exception {
    public RuedasInvalidasException(String msg) {
        super(msg);
    }
}
```

Estamos llamando al superconstructor de la clase Exception con un mensaje, que es el que luego se obtendra.

Con estas excepciones creadas podriamos dejar la clase coche de la siguiente manera:

```
public class Coche {
    String modelo;
    int numeroRuedas;

          public Coche(String modelo, int numeroRuedas) throws RuedaInvalidasException,
ModeloInvalidoException {
        setModelo(modelo);
        setNumeroRuedas(numeroRuedas);
    }

    public void setModelo(String modelo) throws ModeloInvalidoException {
        if (modelo.length() > 50)
            throws new ModeloInvalidoException("El modelo no puede tener mas de 50 caracteres");

        this.modelo = modelo;
    }

    public void setNumeroRuedas(int numeroRuedas) throws RuedaInvalidasException {
        if (numeroRuedas != 3 && numeroRuedas != 4)
            throw new RuedaInvalidasException("El numero de ruedas debe ser 3 o 4");
```

```
        this.numeroRuedas = numeroRuedas;
    }
}
```