Gestión de errores

Qué hacer cuando se pueden producir errores…

```
class EjemploSinControlDeErrores
{
  public static void main(String args[])
  {
    mostrarEntero (args, 0);
  }

  public static void mostrarEntero
                     (String args[], int n)
  {
    System.out.println( "Entero: "
                        + obtenerEntero(args,0) );
  }

  public static int obtenerEntero
                    (String args[], int n)
  {
    return Integer.parseInt(args[n]);
  }
}
```

Idea inicial: Evitar los errores antes de que se produzcan

- Añadir comprobaciones **antes** de operaciones “peligrosas”
- No hacer nada que pueda fallar (algo imposible)

Por tanto, es necesario detectar cuándo se produce un error para tomar las medidas oportunas en cada momento…

Solución 1: Códigos de error

- Añadir comprobaciones **después** de operaciones “falibles”
- Si se detecta algún error, devolver algún tipo de código de error.

```
class EjemploConControlDeErrores
{
  public static void main(String args[]) {
    mostrarEntero (args, 0);
  }

  public static int mostrarEntero
                    (String args[], int n)
  {
    int    i      = obtenerEntero(args,n);
    int    error  = 0;
    String salida = null;

    if (i%2 == 0)
       i = i/2;
    else
       error = -1; // Error en los argumentos

    if (error == 0) {
       if ( Runtime.getRuntime().freeMemory() > (8+10)*2 )
          error = -2; // Memoria insuficiente
    }

    if (error == 0) {
       salida = "Entero: " + i;

       if (salida == null)
          error = -3; // Error al crear la salida
    }

    if (error == 0) {
       System.out.println(salida);

       if (System.out.checkError())
          error = -4; // Error al mostrar la salida
    }

    return error;
  }
```

```
  public static int obtenerEntero
                    (String args[], int n)
  {
    int error = 0;

    if (args == null || args.length == 0)
       error = -1; // Vector inexistente

    if (error == 0) {
       if ((n<0) || (n>=args.length))
          error = -3;
    }

    if (error == 0) {
       if (!comprobarEntero(args[n]))
           error = -5;
    }

    // Devuelve 2*entero, -1, -3 o -5

    if (error == 0)
       return 2*Integer.parseInt(args[n]);
    else
       return error;
  }

  public static boolean comprobarEntero
                        (String entero)
  {
    ...
  }
}
```

Inconvenientes

- ß Demasiadas comprobaciones
- ß Código excesivamente enrevesado (confuso y propenso a errores)

Solución 2: Excepciones

Ya que los errores son inevitables,
las excepciones nos proporcionan una estructura de control que permite implementar los “casos normales” con facilidad
y tratar separadamente los “casos excepcionales”

```
class EjemploConExcepciones
{
  public static void main(String args[])
  {
    try {
        mostrarEntero (args, 0);
    } catch (Exception error) {
        // Casos excepcionales...
    }
  }

  // Casos normales...

  public static void mostrarEntero
                     (String args[], int n)
  {
    System.out.println( "Entero: "
                        + obtenerEntero(args,0) );
  }

  public static int obtenerEntero
                    (String args[], int n)
  {
    return Integer.parseInt(args[n]);
  }
}
```

Además, las excepciones nos permitirán mantener información acerca de lo que falló (tipo de error, detalles relevantes y lugar en el que se produjo el error) y enviar esta información al método que queramos que se encargue de tratar el problema, todo esto sin interferir con el funcionamiento normal del programa (p.ej. sentencias `return`).