Excepciones

Excepciones

A menudo, en los programas se producen *situaciones excepcionales* o inesperadas.

- Esto es frecuente en la entrada/salida: el programa pide un entero y el usuario introduce una cadena de texto que no representa un entero

```
// lee de teclado un entero del intervalo [min,max]
// repite la peticion si el entero no está en ese rango
static int pideEntero(string texto, int min, int max){
  int dato = -1;
  while (dato<min || dato>max){
    Console.Write ("{0} [{1},{2}]: ", texto, min, max);
    dato = int.Parse (Console.ReadLine ());
  };
  return dato;
}
```

- Si $[min, max] = [0, 100]$ y ponemos -4, solicita de nuevo el dato
- Y si introducimos la cadena `doce`?

Si escribimos `doce` ⇝ `int.Parse("doce")` produce un error.

```
Unhandled Exception:
System.FormatException: Input string was not in a correct format.
  at System.Number.StringToNumber (System.String str, NumberStyles options, ...
  at System.Number.ParseInt32 (System.String s, NumberStyles style,  ...
  at System.Int32.Parse (System.String s) ...
  at excepciones.MainClass.pideEntero (System.String texto, ...
  at excepciones.MainClass.Main () [0x0000c] in .../excepciones/Program.cs:13
[ERROR] FATAL UNHANDLED EXCEPTION: System.FormatException: Input string was not in a correct format.
```

⇝ `int.Parse` no *sabe* cómo convertir el string `"doce"` al tipo `int` ⇝ lanza una excepción, i.e., *informa* de que ha ocurrido una circunstancia excepcional.

... y además el programa termina abruptamente

... pero las excepciones pueden capturarse y retomar el control de la situación

La construcción `try-catch`

Rehacemos el código:

```
static int pideEntero2(string texto, int min, int max){
  int dato = -1;
  while (dato<min || dato>max){
    Console.Write ("{0} [{1},{2}]: ", texto, min, max);
    try { // intentamos la conversion
      dato = int.Parse (Console.ReadLine ());
    }
    catch { // si algo no va bien
      Console.WriteLine("Debes introducir un valor entero");
    }
  }
  return dato;
}
```

Esto se denomina a veces *control dinámico de errores* y está relacionado con la `programación defensiva`.

- En el bloques `try` se pueden escribir tantas instrucciones como se quiera: se capturará cualquier excepción producida por ellas en.
- En el bloque `catch` también pueden escribirse tantas instrucciones como se desee y se ejecutarán en caso de excepción.
- Se pueden anidar los `try-catch`

```
try
{
// Las excepciones de este bloque se capturan el catch externo
  try
  {
    // Las excepciones de este bloque se capturan el catch interno
  }
  catch // excepciones internas
  { ... }
  // Las excepciones de este bloque se capturan el catch externo
}
catch // excepciones externas
{ ... }
```


Refinando el `catch`

El `catch` puede capturar el *tipo de excepción* que se produce (en realidad una excepción es un o `objeto` de la clase `Exception`).

- Esto permite obtener más información sobre la excepción (sin detener el programa)

```
static int pideEntero3(string texto, int min, int max){
  int dato = -1;
  while (dato<min || dato>max){
    Console.Write ("{0} [{1},{2}]: ", texto, min, max);
    try {
      dato = int.Parse (Console.ReadLine ());
    }
    catch (Exception e){
      Console.WriteLine (e);
      Console.WriteLine("Debes introducir un valor entero");
    }
  }
  return dato;
}
```

Creando nuestras propias excepciones

El programador también puede lanzar excepciones de la forma:

```
throw new Exception("Algo va muy mal");
```

Esta excepción puede capturarse y manejarse como las anteriores con `try-catch`.

- Las excepciones deben utilizarse en situaciones críticas, cuando el programa no puede continuar normalmente: no deben lanzarse en situaciones que puedan solucionarse fácilmente pidiendo otro valor o dando un simple mensaje de error.