

Tema 14

Módulos

Concepto de módulo

Un módulo es un fragmento de programa desarrollado en forma independiente. El concepto de módulo, por tanto, está íntimamente ligado a la idea de abstracción. Un módulo debe definir un elemento abstracto (o varios relacionados entre sí) y debe ser usado desde fuera con sólo saber *qué hace* el módulo, pero sin necesidad de conocer *cómo lo hace*.

- **Especificación y realización:** La especificación de un módulo que contenga la definición de una serie de elementos abstractos consistirá, fundamentalmente, en el conjunto de las especificaciones de cada uno de ellos, por separado, más una indicación de los posibles efectos de uno sobre otros cuando se usan en forma combinada.

La realización del módulo consistirá en la realización de cada uno de los elementos abstractos contenidos en dicho módulo.

Referida a los módulos, la ocultación consiste en que el programa que usa un elemento de un módulo sólo tiene visible la información de la interfaz, pero no la de la realización.

- **Compilación separada:** Los lenguajes de programación que permiten programar usando módulos pueden emplear diversas técnicas para definirlos e invocar los elementos definidos en ellos.
 - **Compilación separada:** el programa está formado por varios ficheros fuente, cada uno de los cuales se compila por separado.
 - **Compilación segura:** Al compilar un fichero fuente el compilador comprueba que el uso de elementos de otros módulos es consistente con la interfaz.
 - **Ocultación:** Al compilar un fichero fuente el compilador no usa información de los detalles de realización de elementos de otros módulos.

- **Descomposición modular:** La posibilidad de compilar módulos en forma separada permite repartir el trabajo de desarrollo de un programa, a base de realizar su *descomposición modular*.

Para que la descomposición en módulos sea adecuada, desde el punto de vista de la ingeniería de software, conviene que los módulos resulten tan independientes unos de otros como sea posible. Esta independencia se analiza según dos criterios:

- El acoplamiento entre módulos indica cuántos elementos distintos o características de uno o varios módulos han de ser tenidos en cuenta a la vez al usar un módulo desde otro. Este acoplamiento debe verse reducido al mínimo.
- La cohesión indica el grado de relación que existe entre los distintos elementos de un mismo módulo, y debe ser lo mayor posible. Esto quiere decir que dos elementos íntimamente relacionados deberían ser definidos en el mismo módulo.

Módulos en Modula-2

Un programa descompuesto en módulos se describe como un conjunto de ficheros fuentes relacionados entre sí, y que pueden compilarse por separado. Cada fichero fuente constituye así una *unidad de compilación*.

- **Módulo principal:** Un programa se puede descomponer en varios módulos. Uno de estos módulos ha de ser el programa principal o *módulo principal*. La ejecución del programa completo equivale a la ejecución del módulo principal.
- **Módulos de definición:** Las dos unidades de compilación de un módulo corresponden a la especificación (o interfaz) y a la realización (o implementación). La especificación de un módulo se escribe como *módulo de definición* con el siguiente formato:

```
DEFINITION MODULE Nombre;
  ..lista de importaciones..
  ..definiciones..
END Nombre.
```

- **Módulos de implementación:** Contienen la realización de un módulo. Esta realización tiene exactamente la misma forma que un módulo principal, precedido de la palabra clave `IMPLEMENTATION`:

```
IMPLEMENTATION MODULE Nombre;
  ..lista de importaciones..
  ..definiciones..
[ BEGIN
  ..sentencias de inicialización.. ]
END Nombre.
```

Las sentencias ejecutables que figuran al final del módulo son opcionales (si no hay sentencias se suprime también la palabra `BEGIN`). Estas sentencias se ejecutan automáticamente al comienzo de la aplicación, antes de que se ejecute el programa principal, y antes de que se ejecute la inicialización de cualquier otro módulo que utilice este módulo.

- **Uso de módulos:** Para usar elementos de otros módulos hay que importarlos. La importación se realiza al comienzo del programa (o módulo) que desea usarlos. Hay dos formatos:

1. Importación de elementos sueltos:

```
FROM módulo IMPORT ListaDeNombres;
```

Con esta forma de importación los elementos importados se usan directamente por su nombre, tal como si hubieran sido definidos directamente en el programa que los usa.

2. Importación de módulos completos:

```
IMPORT módulo;
```

Con esta forma de importación se puede usar cualquier elemento definido en ese módulo, pero no directamente por su nombre, sino designándolo mediante la combinación:

```
módulo.NombreDeElemento
```

Esta notación se llama *nombre cualificado*, y tiene una apariencia similar a la referencia a campos de un registro.

Tipos abstractos de datos

Es la agrupación de una *colección de valores* y una *colección de operaciones* de manipulación.

Realización de tipos abstractos en Modula-2

- **Definición de tipos abstractos como módulos:** Podríamos decir que la interfaz del módulo equivale a la especificación del tipo, *abstracto*, de datos, y que la realización del módulo establece su representación interna.

- **Tipos opacos:** Para asegurarse de que a los datos de un tipo abstracto se les aplican únicamente las operaciones que se han definido expresamente para ellos, es posible usar un artificio que permite definir los llamados *tipos opacos*.

Un tipo opaco se define en un módulo de definición escribiendo sólo la parte inicial de una definición de tipo, en la forma:

```
TYPE NombreDelTipo;
```

De esta manera es imposible aplicar ninguna construcción del lenguaje a los valores de ese tipo, salvo las siguientes:

- Un valor del tipo opaco puede asignarse a una variable del mismo tipo.
- Los valores o variables de ese tipo pueden pasarse como argumentos.
- Dos valores del mismo tipo pueden compararse para ver si son iguales.
- **Datos encapsulados:** Cuando en un programa sólo hay que manejar una única variable de ese tipo, se puede conseguir fácilmente una ocultación total sin necesidad de recurrir a tipos opacos, si dicha variable única es interna (y anónima) al módulo en que se desarrolla el tipo abstracto.

Desarrollo modular basado en abstracciones

La técnica de programación estructurada, basada en refinamientos sucesivos, puede ampliarse para contemplar la descomposición modular de un programa.

- **Abstracciones para desarrollo modular:** Se distinguen dos tipos de abstracciones:
 - **Abstracciones funcionales:** Se aplican a las expresiones o acciones con independencia de que sean una función propiamente dicha o un procedimiento que realiza una acción.
 - **Abstracciones de datos:** Se aplican a los *tipos abstractos de datos* y a los *datos abstratos encapsulados*.
- **Desarrollo por refinamiento basado en abstracciones:** Las técnicas de refinamiento sucesivo pueden ampliarse con las abstracciones funcionales y las abstracciones de datos en módulos separados.
- **Reutilización de módulos:** Los módulos que definen abstracciones relacionadas entre sí pueden agruparse en una biblioteca que se pone a disposición de quienes desarrollan aplicaciones en un campo determinado.