Programación III

I.T.Informática de Sistemas

Introducción a la POO.

Prof. Félix Prieto Arambillet
Departamento de Informática
Universidad de Valladolid
Curso 2003/2004

 ©–FÉLIX 2003

Factores externos de calidad

- Correcto
- Robusto
- Compatible
- Extensible
- Reutilizable
- ...

Cinco criterios de modularidad.

- Facilidad de descomposición.
- Facilidad de recombinación.
- Facilidad de comprensión.
- Continuidad.
- Protección.

Cinco reglas de modularidad

- Correspondencia directa.
- Pocos interfaces.
- Interfaces pequeños.
- Interfaces explícitos.
- Ocultamiento de información.

Cinco principios de modularidad

- Principio de unidad lingüística.
- Principio de autodocumentación.
- Principio de acceso uniforme.
- Principio de abierto-cerrado.
- Principio de elección única.

Reutilización del Software

- Producción en masa de componentes software.
- Catálogos de módulos.
- Desarrollo a partir de piezas.

Un problema frecuente

Dado un elemento $\mathtt{x}$ de algún tipo, y un conjunto $\mathtt{t}$ de elementos del mismo tipo, se trata de construir un programa buscar que determine si $\mathtt{x}$ está o no en $\mathtt{t}$.

Dificultades de la reutilización de software

- No se conoce su existencia.
- Demasiado caro.
- Inaccesible.
- Reservas psicológicas.
- Herramientas inadecuadas.
- Noción inadecuada de módulo.

Formas de reutilización

- Reutilización del código fuente.
- Reutilización del personal informático.
- Reutilización de diseños genéricos.

Esquema genérico

```
buscar(X:ELEMENTO, t: ESTRUCTURA_DE_ELEMENTO) : booleano
   pos: POSICION
inicio
   pos := POSICION_INICIAL (x,t);
   mientras no AGOTADA (pos, t) y no ENCONTRADO(pos, x, t) hacer
      pos := SIGUIENTE(pos,x,t)
   fin_mientras
   devolver no AGOTADA(pos, t)
fin
```

Requisitos de reutilización

- Variación de tipos.
- Variación de estructuras de datos y algoritmos.
- Posibilidad de agrupar funciones.
- Independencia de la representación.
- Pautas comunes a subgrupos.

Soluciones propuestas

- Creación de bibliotecas de funciones.
- Uso de módulos empaquetados.
- Sobrecarga de operadores.
- Generalización (módulos abstractos).

Creación de bibliotecas de funciones

- Funciona con problemas individuales:
  - ◇ de especificación sencilla
  - ◇ claramente diferenciados
  - ◇ sin estructuras de datos complejas
- Problemas:
  - ◇ Función enorme que hay que recompilar.
  - ◇ Colección de funciones muy parecidas.
  - ◇ Estructuras de datos dispersas.

Módulos empaquetados

- El módulo incluiría:
  - ◇ Definición de un tipo.
  - ◇ Operaciones relacionadas.
- Resuelve el agrupamiento de funciones.
- No resuelve las dificultades con problemas generales o muchos problemas relacionados.

Sobrecarga

- El mismo nombre de función para varios tipos de datos.
- Es una ventaja para los programadores de módulos cliente.
- Avance en el aspecto de variación de tipos.

Generalización

- Permite definir módulos genéricos.
- Substituyendo parámetros formales se obtienen instancias del módulo genérico.
- Proporciona mayor comodidad a los programadores de módulos servidores.
- Herramienta para trabajar con distintas estructuras de datos.

Problemas sin resolver

- Pautas comunes con subgrupos.
  - ◇ Dos niveles de módulos.
  - ◇ Sólo un nivel utilizable y no modificable.
- Independencia de la representación.
  - ◇ Un módulo genérico no es utilizable en sí mismo sino a través de sus instancias.
  - ◇ Cada llamada a una función se refiere a una única versión de la operación, a pesar de la sobrecarga.

El siguiente paso

Plantear una función **buscar(x,t)** que signifique:

> Buscar el elemento x en t, usando el algoritmo apropiado, para cualquier estructura de datos t, sea la que sea, en tiempo de ejecución.

Orientación a procesos

- No tiene en cuenta la orientación evolutiva del software.
- La noción misma de un sistema como un proceso único es cuestionable.
- Se olvida con frecuencia la estructura de los datos.
- El diseño "top-down" no promueve la reutilización.

Orientación a objetos

- Cómo encontrar los objetos.
- Cómo describir los objetos.
- Cómo describir las relaciones y aspectos comunes a objetos.
- Cómo utilizar los objetos para estructurar programas.

Tipos abstractos de datos

tipos

```
PILA[X]
```

funciones

```
vacia: PILA[X] --> BOOLEANO
nueva: --> PILA[X]
apilar: X x PILA[X] --> PILA[X]
extraer: PILA[X] --> PILA[X]
cima: PILA[X] --> X
```

precondiciones

```
pre extraer(p: PILA[X]) = (no vacía(p))
pre cima(p:PILA[X]) = (no vacía(p))
```

axiomas

```
∀  x: X, p: PILA[X]
  vacía(nueva())
  no vacía(apilar(x,p))
  cima(apilar(x,p))=x
  extraer(apilar(x,p))=p
```

Clases y TAD

- El diseño orientado a objetos construye sistemas informáticos como colecciones estructuradas de implementaciones de TAD.
- Los módulos representan una implementación de un TAD, no el TAD en sí mismo.

Clases y TAD

```
Clase PILA[X]
   num_elementos : ENTERO;
   función vacía: BOOLEANO
      inicio
      ...
      fin;
   ...
   función cima : X
      (precondición: pila no vacía)
      inicio
      ...
      fin;
fin clase
```

Clases y herencia

- Las clases se diseñan como unidades interesantes y útiles en sí mismas.
- Relaciones importantes entre clases:
  - ◇ Cliente de (proveedora de)
  - ◇ Descendiente o heredera de (Ascendente o ancestro de)

```
Clase E_SECUENCIAL[T]
Clase TABLA[T] desciende de E_SECUENCIAL[T]
Clase LISTA[T] desciende de E_SECUENCIAL[T]
```

Polimorfismo

- Posibilidad de referirse en tiempo de ejecución a instancias de varias clases.
- Limitado por la herencia. Sólo a clases descendientes.
- Sistema de tipos más flexible.

Ligadura dinámica

Permite seleccionar la implementación concreta de una función según sea el tipo de la entidad en el momento de la ejecución.

```
Clase POLIGONO ...(incluye dibuja)
Clase RECTANGULO desciende de POLIGONO ...
          (incluye dibuja)
p: POLIGONO
r: RECTANGULO
...
p:=r;
p.dibuja
```

Requisitos de la OO según Meyer

- Estructura modular basada en datos.
- Abstracción de datos.
- Gestión de memoria automática.
- Clases.
- Herencia.
- Polimorfismo.
- Herencia múltiple.