Tema 5: Tipos Abstractos de Datos (TADs)

Tecnología de la Programación

Índice

1. Introducción
2. Definición de Tipos Abstractos de Datos
3. Diseño modular

Índice

1. **Introducción**
2. Definición de Tipos Abstractos de Datos
3. Diseño modular

Introducción

Tipo Abstracto de Datos (TADs):

- Tipo de datos (TDs):
  - Concepto introducido a mediados de los 70
  - Consta de:
    - Conjunto de **valores**
    - Conjunto de **operaciones**
    - **Propiedades** que determinan su comportamiento
- Abstracto:
  - Destacar detalles relevantes
  - Ignorar detalles irrelevantes
  - Caso de los tipos de datos:
    - Olvidarse de cómo están hechas las cosas
    - Centrarse en cómo funcionan (es decir, lo que hacen)

Introducción

Uso de la abstracción en programación:

- Ordenador lógico -- físico
- Lenguajes de alto nivel -- máquina
- Subalgoritmos -- instrucciones virtuales
- TDs definidos por el programador -- modelo del mundo real
  - Ejemplos:
    - `int` $\rightarrow \mathbb{Z}$
    - `float` $\rightarrow \mathbb{R}$
    - Sin conocer detalles de implementación (simplemente conociendo su comportamiento) sabemos usarlos
    - TDs predefinidos son en realidad TADs que nos dan ya definidos

Introducción

Objetivo:

- Simular en el ordenador modelos del mundo real

Equiparar tipos predefinidos en lenguajes de programación con los definidos por el usuario. Para ello, se exigen dos propiedades:

- Privacidad de la representación:
  - Usuarios no conocen como los TDs están representados en memoria
- Protección:
  - Usuarios sólo pueden usar propiedades y métodos previstos en la especificación

Punto de partida

- Tenemos en la cabeza un modelo de lo que queremos simular en el ordenador
- Puede ocurrir que el lenguaje que utilicemos no contenga un TD que lo simule
- Por lo que será necesario definir un TAD
- Ejemplos:

Nuestra cabeza	Ordenador
$\mathbb{Z}$	`int`
$\mathbb{R}$	`float`
$\mathbb{C}$	`???`

Índice

1. Introducción
2. **Definición de Tipos Abstractos de Datos**
3. Diseño modular

TADs

Definición: Un TAD es un conjunto de valores (*dominio*) junto con un conjunto de operaciones que se pueden aplicar sobre ellos, y que verifican una serie de propiedades que determinan su comportamiento

Un TAD viene caracterizado por:

- Dominio
- Operaciones
- Propiedades

Ejemplo de TAD

En C++ el tipo `int`:

- Dominio ≡ [minint,maxint]
- Operadores ≡ +, -, *, /
- Propiedades ≡ a+b = b+a, …

Definición de un TAD

1. Especificación:
    - Establece el comportamiento del TAD (valores, operaciones, y propiedades)
    - Única que conoce el usuario del TAD
    - Propiedades deseables: precisa, general, legible, y no ambigua
    - Independiente del hardware y del software
2. Implementación:
    - Determina representación de los valores del tipo y codifica las operaciones en base a esa representación usando un lenguaje de programación
    - Propiedades deseables: estructurada, eficiente y legible
    - Una misma especificación puede tener distintas implementaciones

Desarrollo de un TAD

1. Especificación:
    - Única que conoce el usuario del TAD
    - Contiene:
        - Nombre del TAD
        - Especificación de las operaciones:
            - Sintaxis: cabecera
            - Semántica: descripción del comportamiento
2. Implementación:
    - Contiene:
        - Representación → usando tipos conocidos:
            - Dominio: cómo representar los valores
            - Interpretación: paso del dominio al modelo
        - Implementación de las operaciones (supeditada a la representación)

Ejemplo de uso de un TAD

Diseñar algoritmo que calcule la suma de una secuencia de números complejos introducidos por teclado (terminar con 0+0i) y escribir el resultado

Suponer definido el TAD complejo:

- Dominio: números complejos cuyas partes real e imaginaria son reales
- Operaciones:
  - Constructores → crear complejo a partir de dos reales
  - Accesores → obtener parte real e imaginaria
  - Sumar
- Propiedades: típicas de los números complejos

El TAD Complejo

- Especificación
- Uso
- Implementaciones

Índice

1. Introducción
2. Definición de Tipos Abstractos de Datos
3. **Diseño modular**

Diseño modular

Generalización del diseño descendente:

- Diseño descendente → Descomposición en acciones (verbos) → **Programa = conjunto de subprogramas**
- Diseño modular → Descomposición en objetos (nombres) → **Programa = conjunto de TADs**

Idea: Módulo = TAD (especificación + implementación)

Diseño modular

Ejemplo: Gestión de un banco, operación: listado de movimientos en un día

- Descendente: listar, dar alta, realizar movimiento, …
- Modular: cuenta, cliente, movimiento, …

Ejemplo: Simular partida de cartas

- Descendente: jugar, barajar, …
- Modular: jugada, baraja, carta, partida, …

Diseño modular

- Cada TAD se encapsula en un módulo
- **Módulo** = unidad de programa que agrupa o encapsula un conjunto de definiciones de objetos permitiendo ocultar todos los detalles internos relativos a su representación o ejecución
- **Programa** = colección estructura de implementaciones de TADs
- Declaración de un módulo:
  - Encabezamiento
  - Parte pública → interfaz
  - Parte privada → implementación

Diseño modular

Propiedades de la programación modular:

- Abstracción
- Corrección
- Eficiencia
- Legibilidad
- Modificabilidad
- Organización
- Reusabilidad
- Seguridad