Tipos Abstractos de Datos

Tema 1. Estructuras de Datos y de la Información

Tipos de Datos

- Un Tipo de Datos es una *colección de valores*
- Han sido estudiados los tipos de datos que implementan lenguajes como ADA o Java (Boolean, Integer, Character...)
- Estos tipos son conocidos como “tipos simples”
- Estos tipos pueden ser utilizados en nuestros programas sin necesidad de que los detalles sobre su implementación sean conocidos

Tipos Abstractos de Datos (TAD) (I)

- Podemos encontrar varias definiciones para el concepto de Tipo Abstracto de Datos (TAD)

TAD: Conjunto de Operaciones. ***Weiss, Data Structures andAlgorithms.*** **p.46.**

TAD: Modelo matemático con una serie de operaciones definidas en ese modelo. ***Aho, Hopcroft, Ullman, Data Structures and Algorithms.*** **p.11.**

TAD: Tipo de datos definido de forma única mediante un tipo y un conjunto de operaciones definidas sobre el tipo. ***Hernández, Lázaro, Dormido, Ros. Estructuras de Datos y Algoritmos.*** **p.3.**

Tipos Abstractos de Datos (TAD) (II)

- Un Tipo Abstracto de Datos es una abstracción donde se encuentran encapsulados los ***estados potenciales*** en los que se puede encontrar una entidad de ese tipo y las ***operaciones*** que pueden realizarse sobre ella.
- Abstraer: Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.

Tipos Abstractos de Datos (TAD) (II)

- Como se ha mencionado, se trata de una abstracción. No se incluyen detalles sobre la implementación de las operaciones.
- Los TAD son independientes por completo de la implementación.

Estructuras de Datos

- En muchos textos, pueden encontrarse confundidos los términos TAD y Tipo de Datos, así como TAD y Estructura de Datos.
- **Estructura de Datos:** Conjunto de variables que se encuentran relacionadas.
- Con Estructura de Datos, por tanto, nos referimos a la *implementación física* de un TAD.

Atributos de un TAD

- El encapsulamiento y la ocultación de información son atributos internos del diseño.
- Un TAD tiene estas propiedades:
  - **Encapsulamiento:** La información referente a la definición del tipo y todas las operaciones que pueden realizarse sobre el mismo se encuentran en el mismo lugar.
  - **Ocultación de Información:** La información acerca de la implementación se encuentra oculta al usuario.

Implementación de un TAD

- Es importante comprender la diferencia entre un TAD y su implementación
- Las implementaciones no dejan de ser importantes, y su elección es crítica
- Al final, el usuario no debe preocuparse de cómo está implementado un TAD. Su única preocupación debe ser el uso del mismo

Implementación de un TAD (II)

- ¿Cómo debe implementarse un TAD?
- Deben considerarse detalles acerca de la complejidad espacial de las estructuras y temporal de las operaciones
- Preguntas que debe formularse el programador:
  - ¿Cómo será la estructura de datos? ¿Cómo crecerá?
  - Según lo anterior y otras consideraciones ¿cuál será el coste de una implementación u otra para cada operación?

TAD: Diseño e implementación

- Debido a todo lo expuesto, el diseñador de un TAD debe enfrentarse a tres pasos bien distintos, pero íntimamente relacionados:

1. **Análisis de datos y operaciones**
2. **Elección del TAD**
3. **Elección de la implementación**

Declaración de un TAD: Especificación Algebraica

- Una especificación algebraica de un TAD tiene dos componentes:

1. **Signatura** (Sintaxis): Se compone de
   a) Definición de los posibles valores del tipo
   b) Operaciones definidas
2. **Axiomas** (Semántica): Relaciones y restricciones que se establecen sobre el modelo

Ejemplo 1: TAD Booleano

TAD Booleano;

SIGNATURA
VALORES
BOOLEAN={TRUE,FALSE}
OPERACIONES
INIC:BOOLEAN → BOOLEAN
NOT:BOOLEAN → BOOLEAN
OR: BOOLEAN x BOOLEAN → BOOLEAN
AND: BOOLEAN x BOOLEAN → BOOLEAN

Ejemplo 1: TAD Booleano (II)

AXIOMAS

INIC(p) = p
NOT(TRUE) = FALSE
NOT(NOT(p)) = p
p OR NOT(p) = TRUE
p OR p = p
p AND NOT(p) = FALSE
p AND p = p

Ejemplo 2: TAD Número_Natural

TAD Número_Natural;
UTILIZA Booleano;
SIGNATURA
VALORES
NAT
OPERACIONES
INIC: NAT → NAT
CERO: → NAT
ESCERO: NAT → BOOLEAN
SUCC: NAT → NAT
SUM: NAT x NAT → NAT
IGUALES: NAT x NAT → BOOLEAN

Ejemplo 2: TAD Número_Natural (II)

AXIOMAS

∀ X, Y ∈ NAT:
INIC(X) = X
CERO() = 0
ESCERO(CERO) = TRUE
ESCERO(SUCC(X)) = FALSE
SUMA(CERO,X) = X
SUMA(SUCC(X),Y) = SUCC(SUMA(X,Y))
SUCC(X) = SUMA(X,1)
IGUAL(X,CERO) = IF ESCERO(X) THEN TRUE
ELSE FALSE
IGUAL(CERO,SUCC(X)) = FALSE
IGUAL(SUCC(X),SUCC(Y)) = IGUAL(X,Y)

Construcción de TAD en Java: Clases y Objetos

- Java proporciona mecanismos para encapsular y ocultar la información.
- Encapsulamiento de datos en clases y objetos.
- Una clase constituye un nuevo tipo, y los objetos de esta clase serán instancias (entidades físicas) de este tipo.

Construcción de TAD en Java

- Esta construcción (la clase) permite, por tanto, encapsular datos y operaciones sobre los mismos, por lo que se revela idónea para construir TADs.
- Cada clase Java se corresponde con un fichero, que representa la declaración e implementación de un TAD.
- En el archivo que contenga el TAD deben encontrarse representados los atributos de la clase (privados), el nombre y forma de las operaciones que se exportan (su interfaz), y su implementación.

Ejemplo de implementación: TAD conjunto

```
public class Conjunto {
    private char contenedor[ ];
    private int tam;
    public Conjunto(){
        contenedor = new char[256];
        tam = 0;
    }
    public Conjunto(Conjunto S){
        contenedor = new char[256];
        for (int i = 0; i < S.length; i++)
            contenedor[i] = S.contenedor[i];
        tam = S.tam;
    }
```

Ejemplo de implementación: TAD conjunto

```
public int posicion (char c){
    int i = 0;
    while (i < tam && contenedor[i] != c)
        i++;
    return i == tam ? -1 : i;
}
public boolean contiene (char c){
    return posicion(c) >= 0;
}
public boolean vacio (){
    return tam == 0;
}
public boolean lleno (){
    return tam == 256;
}
```

Ejemplo de implementación: TAD conjunto

```
public void insertar (char c){
    if (!contiene(c)){
        contenedor[tam] = c;
        tam++;
    }
}

public void eliminar(char c){
    int pos = posicion(c);
    if (pos >= 0){
        for (int i = pos+1; i<contenedor.length; i++)
            contenedor[i-1] = contenedor[i];
        tam--;
    }
}
```

Ejemplo de implementación: TAD conjunto

```
public Conjunto union (Conjunto s){
    Conunto res = new Conjunto(s);
    for (int i = 0; i < this.tam; i++)
        res.insertar(this.contenedor[i]);
    return res;
}

public Conjunto interseccion (Conjunto s){









}
```

Contenedores homogéneos genéricos: Generics

- Contenedor Homogéneo: Todos los registros contienen el mismo tipo de datos (e.g. array).
- Contenedor Homogéneo Genérico: El tipo de datos se decide en tiempo de compilación (e.g. array).
- En Java pueden crearse contenedores homogéneos genéricos de manera eficiente utilizando *generics*.

Contenedores homogéneos genéricos: Generics (II)

- Una clase *generic* recibe parámetros para su conformación como tal.
- En tiempo de compilación se creará el código real de la clase.
- Los parámetros pueden ser tipos. En el caso en el que se esté construyendo un contenedor homogéneo genérico, el tipo contenido siempre es un parámetro.

Contenedores homogéneos genéricos: Generics (III)

- Ejemplo de contenedor homogéneo genérico: Conjunto de cualquier tipo.
- Para declarar una clase (o función) como generic, debemos incluir en la declaración la siguiente secuencia:

```
<modificador> class <T>
<declaración estándar>
```

- <**T**> indica que vamos a declarar una clase genérica con un tipo paramétrico T.

Ejemplo de implementación: TAD conjunto genérico

```
public class Conjunto <T>{
    private T contenedor[ ];
    private int tam;
    public Conjunto(){
        contenedor = (T[]) new Object[256];
        tam = 0;
    }
    public Conjunto(Conjunto<T> S){
        contenedor = (T[]) new Object[256];
        for (int i = 0; i < S.length; i++)
            contenedor[i] = S.contenedor[i];
        tam = S.tam;
    }
```

Ejemplo de implementación: TAD conjunto genérico

```
    public int posicion (T c){
        int i = 0;
        while (i < tam && contenedor[i] != c)
            i++;
        return i == tam ? -1 : i;
    }
    public void contiene (T c){
        return posicion(c) >= 0;
    }
    public boolean vacio (){
        return tam == 0;
    }
    public boolean lleno (){
        return tam == 256;
    }
```

Ejemplo de implementación: TAD conjunto genérico

```
    public void insertar (T c){
        if (!contiene(c)){
            contenedor[tam] = c;
            tam++;
        }
    }

    public void eliminar (T c){
        int pos = posicion(c);
        if (pos >= 0){
            for (int i = pos+1; i<contenedor.length; i++)
                contenedor[i-1] = contenedor[i];
            tam--;
        }
    }
```

Ejemplo de implementación: TAD conjunto genérico

```
public static void main(String args[]){

    Conjunto<String> a = new Conjunto<String>();

    Conjunto<int> b = new Conjunto<int>(); //Incorrecto

    Conjunto<Integer> b = new Conjunto<Integer>();
}
```

Diseño por contrato (I)

- Cuando se fija un contrato, se desprenden de este las obligaciones y beneficios de cada una de las partes implicadas.
- Siguiendo una metáfora aplicada a software, en el diseño por contrato se establecerán las relaciones entre los procedimientos y sus invocadores.
- El Diseño por Contrato da una visión de la construcción del software como un conjunto de elementos cooperando entre sí.

Diseño por contrato (II)

- El diseño por contrato toma su base de la especificación formal de programas, pero es mucho más pragmático.
- Siguiendo con la metáfora, se establecerán relaciones *Proveedor-Cliente,* siendo estos dos los roles que tomarán los elementos del software.
- Se establecerá un contrato entre las dos partes que especificará obligaciones y beneficios de ambas partes (una obligación de una parte supone un beneficio de la otra).

Diseño por contrato (III)

- Los contratos se especifican usando asertos lógicos de tres tipos:
  - Precondiciones.
  - Postcondiciones.
  - Invariantes (de clase).

Diseño por contrato (IV)

- En lo que se refiere a obligaciones y beneficios, la relación entre cliente y proveedor viene dada por la siguiente tabla:

	Obligaciones	Beneficios
Proveedor	Satisfacer la postcondición	No preocuparse por la precondición
Cliente	Satisfacer la precondición	Obtiene lo establecido en la postcondición

Diseño por contrato (V)

- Invariante de una clase: Predicado que expresa el conjunto de estados de corrección para la clase.
- Un contrato debe incluir, entonces, una precondición, una postcondición y un invariante.
- El no cumplimiento de alguna de las cláusulas del contrato significará un defecto en el software.

Diseño por contrato (VI)

- Utilizaremos excepciones al iniciarse la llamada a una función para verificar la precondición y el invariante, y al final para hacer lo propio con el invariante y la postcondición.
  - Las excepciones de tipo RuntimeException no exigen declarar la cláusula throws en la cabecera del método.
  - Tampoco obligan a invocar al método dentro de un bloque try-catch.