**1. Introducción**

Los Tipos de Datos Abstractos (TDA) son una herramienta fundamental en el mundo de la programación y la informática en general. Permiten encapsular datos y operaciones relacionadas en una sola entidad, facilitando la modularidad, el reúso de código y la abstracción. En este manual, exploraremos el uso de los TDA, su importancia y cómo implementarlos en diferentes contextos.

**2. Definición de Tipos de Datos Abstractos (TDA)**

Un Tipo de Dato Abstracto (TDA) es una abstracción matemática que define un conjunto de valores y un conjunto de operaciones que pueden realizarse sobre esos valores. Es abstracto en el sentido de que no especifica cómo se implementan los datos ni cómo se realizan las operaciones, sino que se enfoca en definir qué operaciones son posibles y cuál es su comportamiento.



**Imagen 1** Ciclo de vida del TDA

**3. Importancia de los TDA**

Los TDA son importantes porque promueven la modularidad y la encapsulación en el diseño de software. Al encapsular datos y operaciones relacionadas en una sola entidad, se simplifica la complejidad del sistema y se facilita su mantenimiento y extensión. Además, los TDA promueven la reutilización de código, ya que una vez definido un TDA, puede ser utilizado en diferentes partes del programa sin necesidad de conocer los detalles de su implementación.

**4. Uso de TDA**

Los TDA se utilizan en una amplia variedad de contextos, desde la programación orientada a objetos hasta la programación funcional. A continuación, se presentan algunas formas comunes en las que se utilizan los TDA:

Uso de TDA en programación: En programación, los TDA se utilizan para modelar conceptos abstractos como listas, conjuntos, pilas, colas, árboles, entre otros. Estos TDA proporcionan una interfaz clara y coherente para interactuar con los datos, independientemente de cómo estén implementados internamente.

Ejemplos de TDA comunes: Algunos ejemplos comunes de TDA incluyen:

* Listas enlazadas
* Árboles binarios
* Colas y pilas
* Conjuntos y diccionarios
* Grafos

En otras palabras, un TAD es un tipo de datos construido por el programador para resolver una determinada situación. Para definir un TAD, el programador debe comenzar por definir las operaciones que se pueden realizar con él, es decir, qué operaciones son relevantes y útiles para operar con las variables pertenecientes al mismo. Esto se conoce como establecer la interfaz del tipo. La interfaz permite al programador utilizar el tipo (qué se puede hacer frente a cómo está hecho).

**5. Implementación de TDA**

La implementación de un TDA puede variar dependiendo del lenguaje de programación y del contexto en el que se utilice. Sin embargo, hay algunas pautas generales que se pueden seguir:

* Definir una interfaz clara que especifique las operaciones que se pueden realizar sobre el TDA.
* Implementar las operaciones utilizando las estructuras de datos y algoritmos adecuados.
* Ocultar los detalles de implementación detrás de la interfaz, utilizando encapsulamiento u otros mecanismos de protección de datos.

Siguiendo con el ejemplo de las fechas, podemos indicar que las operaciones válidas sobre una fecha son, entre otras:

Crear (dia, mes, año: natural): fecha

Incrementar (fechaInicio: fecha; numDias: entero): fecha

Distancia (fechaInicio, fin: fecha): entero

ObtenerMes (f: fecha): natural

***Fecha.h***

**#ifndef FECHA\_H**

**#define FECHA\_H**

**class Fecha {**

**private:**

**int dia, mes, anio;**

**public:**

**Fecha(int d, int m, int a);**

**Fecha incrementar(int numDias);**

**int distancia(Fecha fin); // cambiar la variable periodo de tiempo.**

**int obtenerMes();**

**private:**

**int diasEnMes(int m, int a);**

**int diasDesde1900(int d, int m, int a);**

**};**

***Fecha.cpp***

**#endif**

**#include "Fecha.h"**

**Fecha::Fecha(int d, int m, int a) {**

**dia = d;**

**mes = m;**

**anio = a;**

**}**

**Fecha Fecha::incrementar(int numDias) {**

**dia += numDias;**

**if (dia > diasEnMes(mes, anio)) {**

**dia -= diasEnMes(mes, anio);**

**mes++;**

**if (mes > 12) {**

**mes = 1;**

**anio++;**

**}**

**}**

**return Fecha(dia, mes, anio);**

**}**

**int Fecha::distancia(Fecha fin) {**

**int diasInicio = diasDesde1900(dia, mes, anio);**

**int diasFin = diasDesde1900(fin.dia, fin.mes, fin.anio);**

**return diasFin - diasInicio;**

**}**

**int Fecha::obtenerMes() {**

**return mes;**

**}**

**int Fecha::diasEnMes(int m, int a) {**

**if (m == 2) {**

**if (a % 4 == 0 && (a % 100 != 0 || a % 400 == 0)) {**

**return 29;**

**} else {**

**return 28;**

**}**

**} else if (m == 4 || m == 6 || m == 9 || m == 11) {**

**return 30;**

**} else {**

**return 31;**

**}**

**}**

**int Fecha::diasDesde1900(int d, int m, int a) {**

**int dias = 0;**

**for (int i = 1900; i < a; i++) {**

**if (i % 4 == 0 && (i % 100 != 0 || i % 400 == 0)) {**

**dias += 366;**

**} else {**

**dias += 365;**

**}**

**}**

**for (int i = 1; i < m; i++) {**

**dias += diasEnMes(i, a);**

**}**

**dias += d - 1;**

**return dias;**

}

**6. Conclusiones**

En resumen, los Tipos de Datos Abstractos (TDA) son una herramienta poderosa para la programación y el diseño de software. Al encapsular datos y operaciones relacionadas en una sola entidad, los TDA promueven el modularidad, el reúso de código y la abstracción, facilitando el desarrollo de sistemas complejos.

**7. Referencias**

[1] Abstraction and Specification in Program Development, Barbara Liskov and John Guttag.

[2] Data Structures and Algorithms in Java, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser.