Tipos de datos abstractos

Jorge Mario Londoño Peláez & Varias AI

January 29, 2025

1 Concepto de Tipos de Datos Abstractos

En esta sección, exploraremos los conceptos fundamentales relacionados con los Tipos de Datos Abstractos (Tipo de Dato Abstracto (TDA) - Abstract Data Type). Es crucial entender cómo estos se relacionan con las representaciones de datos a bajo nivel y los tipos de datos primitivos.

- **Representaciones de bajo nivel**: En el nivel más básico, los datos se representan como secuencias de bits (datos binarios). Estas representaciones son directamente interpretadas por el hardware de la computadora.
- **Tipos de datos primitivos**: Son los tipos de datos básicos que los lenguajes de programación ofrecen directamente, y que tienen un soporte nativo en el hardware. Ejemplos comunes incluyen `int` (enteros), `float` (números de punto flotante), `char` (caracteres) y `bool` (booleanos).
- **Tipos de datos abstractos (TDA)**: Los TDA son modelos de datos que se definen en términos de las operaciones que se pueden realizar sobre ellos, en lugar de cómo se almacenan en la memoria. Un TDA encapsula la representación de los datos y las operaciones permitidas, ofreciendo una interfaz clara y bien definida. Los TDA se construyen utilizando tipos de datos primitivos u otros TDA más simples, permitiendo modelar los datos de una manera más cercana a la lógica del problema que se está resolviendo.

2 Beneficios de los TDA

Los Tipos de Datos Abstractos (TDA - Abstract Data Type (Tipo de Dato Abstracto) (ADT)) ofrecen varios beneficios importantes en el desarrollo de software, que mejoran la calidad, la mantenibilidad y la reutilización del código:

- **Alto nivel de abstracción**: Los ADT permiten modelar los datos de una manera que se asemeja mucho a la forma en que se conciben en el problema real. Esto facilita la comprensión del problema y el diseño de la solución, ya que se trabaja con conceptos más cercanos al dominio del problema.
- **Modularidad y Reutilización**: Los ADT fomentan la creación de componentes de software reutilizables. Una vez que se define un ADT, se puede utilizar en diferentes partes de un programa o incluso en diferentes proyectos, lo que reduce la duplicación de código y acelera el desarrollo.

- **Encapsulación y Ocultamiento de la Información**: Los ADT encapsulan la representación interna de los datos y exponen una interfaz bien definida para interactuar con ellos. Esto oculta los detalles de implementación y protege los datos de manipulaciones incorrectas, lo que ayuda a mantener la consistencia y la integridad de los datos.
- **Facilidad de Mantenimiento**: Al separar la interfaz de la implementación, los ADT facilitan el mantenimiento y la evolución del software. Los cambios en la implementación de un ADT no afectan al resto del código, siempre y cuando la interfaz se mantenga igual.

3 Implementación de TDA en la Programación Orientada a Objetos (POO)

En la Programación Orientada a Objetos (Programación Orientada a Objetos (Object Oriented Programming) (POO) - Object Oriented Programming (Programación Orientada a Objetos) (OOP)), los Tipos de Datos Abstractos (TDA - ADT) se implementan utilizando clases y objetos. Esta sección detalla cómo se lleva a cabo esta implementación:

- **Representación del Estado del TDA**: El estado de un ADT se representa mediante las variables de instancia (también conocidas como atributos o campos) dentro de una clase. Estas variables almacenan los datos que definen el estado específico de cada objeto del ADT.
- **Interfaz del TDA (Application Programming Interface (Interfaz de Programación de Aplicaciones) (API))**: La interfaz de un ADT se define a través de los métodos públicos de la clase. Estos métodos son las operaciones que se pueden realizar sobre los objetos del ADT, y permiten interactuar con los datos de manera controlada y segura.
- **Clases como Implementación de TDA**: En OOP, un ADT se implementa como una clase. La estructura del ADT se define por los tipos de datos de las variables de instancia, mientras que el comportamiento se define por los métodos de la clase.
- **Objetos como Instancias de TDA**: Los datos del tipo del ADT se representan como instancias de la clase, es decir, objetos. Cada objeto es una realización concreta del ADT, con su propio estado específico.

4 Ejemplos básicos de TDA

A continuación, se presentan algunos ejemplos básicos de Tipos de Datos Abstractos (ADT) implementados en Python, mostrando cómo se definen las clases y sus métodos para encapsular los datos y las operaciones:

- **TDA Fecha**: Representa una fecha con día, mes y año.

```python
class Fecha:
    def __init__(self, dia, mes, anno):
        self.dia = dia
        self.mes = mes
        self.anno = anno

    def __str__(self):
        return f"{self.dia}/{self.mes}/{self.anno}"
```

- **TDA Contador**: Representa un contador con un nombre y un valor entero.

```python
class Contador:
    def __init__(self, nombre, valor=0):
        self.nombre = nombre
        self.valor = valor

    def incrementar(self):
        self.valor += 1

    def __str__(self):
        return f"{self.nombre}: {self.valor}"
```

- **TDA Punto**: Representa un punto en un plano cartesiano con coordenadas x e y.

```python
class Punto:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def mover(self, dx, dy):
        self.x += dx
        self.y += dy

    def __str__(self):
        return f"({self.x}, {self.y})"
```

5 Prueba unitaria de las operaciones de un ADT

Las pruebas unitarias son una parte fundamental del desarrollo de software, ya que permiten verificar que cada componente del programa (en este caso, las operaciones de un ADT) funciona correctamente de forma aislada. En esta sección, explicaremos cómo implementar pruebas unitarias sencillas utilizando sentencias `assert` en Python, sin necesidad de librerías especializadas.

Las pruebas unitarias se basan en la idea de crear casos de prueba que ejerciten diferentes caminos de ejecución del código y verificar que el comportamiento observado coincide con el comportamiento esperado. Cada caso de prueba debe probar una funcionalidad específica del ADT.

A continuación, se muestra un ejemplo de cómo implementar una prueba unitaria para el método `mover` del ADT `Punto`:

```python
# Prueba unitaria del metodo mover
p = Punto(1, 2)
assert str(p) == "(1, 2)" # verificar el estado inicial
p.mover(3, 4)
assert str(p) == "(4, 6)" # verificar el estado despues de mover
print("Prueba unitaria del metodo mover: OK")
```

En este ejemplo, primero se crea un objeto `Punto` con coordenadas iniciales (1, 2). Luego, se utiliza una sentencia `assert` para verificar que la representación en cadena del punto es la esperada. A continuación, se llama al método `mover` para desplazar el punto y se utiliza otra sentencia `assert` para verificar que las coordenadas del punto se han actualizado correctamente. Si alguna de las aserciones falla, el programa se detendrá con un error. Si todas las aserciones pasan, se imprimirá un mensaje indicando que la prueba unitaria ha sido exitosa.

Este enfoque sencillo permite verificar el comportamiento de las operaciones de un ADT de manera efectiva y sin necesidad de herramientas complejas. Es importante crear suficientes casos de prueba para cubrir todos los posibles escenarios y asegurar la calidad del código.