Tipo de Dato Abstracto

jueves, 21 de octubre de 2021 12:12

- La mayoría de los lenguajes modernos proveen soporte para proveer **abstracciones** de datos y de procesos
- El concepto de **tipo de dato abstracto** permite en una sola abstracción unificar las abstracciones de datos y de proceso en **un solo tipo**
- Permite unificar la representación de los datos y el código que lo manipula
- Oculta los detalles de la implementación
- **Definición**
  - Es un tipo de datos definido por el usuario que satisface dos restricciones:
    - Ocultamiento de Información:
      - Separación de la **interfaz** del tipo definido respecto a la representación de objetos y el código de los operadores (implementación), estando **oculto** para las unidades del programa que lo utilizan
    - Encapsulamiento:
      - Declaración del tipo y los protocolos de las operaciones sobre objetos del tipo están contenidos en una única unidad sintáctica

- **Ventajas**
  - Modularidad:
    - Encapsulamiento de especificación de datos y operaciones en un solo lugar
    - Es conocido por ottas unidades del programa solo por su interfaz, bajando así su carga cognitiva
  - Modificabilidad:
    - Reforzada al proveer interfaces que son independientes de la implementación, dado que puede modificar la implementación del módulo sin afectar el resto del programa
    - Promueve la compilación separada
  - Reusabilidad:
    - Interfaces estándares del módulo permite que su codificación sea reusada por diferentes programas
  - Seguridad:
    - Permite proteger el acceso a detalles de implementación a otras partes del programa

- **Ejemplo**
  - El ***stack*** se puede abstraer mediante el siguiente conjunto de operaciones sobre el tipo *stack:*
    - create*(stack)* -> Construye un nuevo *stack*
    - destroy*(stack)* -> Destruye el *stack*
    - empty(*stack)* -> Verifica que el *stack* esta vacio
    - ***En el PDF está el resto***
  - El cómo se maneja el *stack* no es de importancia para el que usa el *TDA*, ya que esta implementado por el programador

```
class StackClass {
                private int [] stackRef;
                private int maxLen, topIndex;
                public StackClass() { // un constructor
                        stackRef = new int [100];
                        maxLen = 99;
                        topIndex = -1;
                };
                public void push (int num) {…};
                public void pop () {…};
                public int top () {…};
                public boolean empty () {…};
}
```

- **Java**
  - Es similar a *C++*, excepto:
    - Todos los tipos definidos por el usuario son ***clases***
    - Todos los objetos son ***asignados desde el heap*** y accedidos mediante variables de referencia
    - Los miembros de las clases tienen modificadores de control de acceso (***private*** o ***public***)en vez de capsulas
    - Los objetos en el ***private*** solo pueden ser accedidos por el *TDA*
    - Tiene un segundo mecanismo de ámbito llamado ***paquete*** (Es como una carpeta con varios tipos de datos)

- **Python**
  - Provee soporte para definir clases y objetos pero la definición es más relajada:
    - Una clase se declara con la palabra reservada ***class***
    - El constructor y los métodos deben llevar como primer parámetro el argumento ***self***
    - Una instancia de una clase se hace invocando al ***constructor***
      - *miObjeto = miClase(…)*

```
class Stack:
  def __init__(self, max):
    self.maximo = max
    self.stack = []

  def push(self, obj):
    if len(self.stack) < self.maximo:
      self.stack.append(obj)
      print("push", obj)
    else:
      print("stack overflow")

  def pop(self):
    if len(self.stack) == 0:
      print("stack is empty")
    else:
      print("pop", self.stack.pop())
```

- **Tipo de Dato Parametrizado**
  - Permite diseñar un tipo que puede almacenar cualquier tipo de elemento, siendo este constructo de tipos solo relevante para lenguajes estáticos
  - También conocidos como ***clases genéricas***
  - Soportado por *ADA, C++, C#*
  - Java:
    - El usuario puede definir ***clases genéricas*** donde los ***parámetros genéricos*** deben ser ***clases***
    - No pueden almacenar tipos de datos primitivos
    - Evita tener diferentes tipos de estructuras y hacer *castings* para objetos

Tipo de Dato Abstracto Parametrizado: Ejemplo en C++

```
template <class Type>
class Stack {
 private:
  Type  *stackPtr;
  const int maxLen;
  int   topPtr;
 public:
  Stack(int size) // Constructor
  {
   stackPtr = new Type[size];
   maxLen = size - 1;
   topSub = -1;
  }
...
}
```

Instanciación: Stack<int> miIntStack;

Tipo de Dato Abstracto Parametrizado: Ejemplo en Java

```
import java.util.*;

public class Stack2<T> {
 private ArrayList<T> stackRef;
 private int maxLen;
 public Stack2(int size) {
  stackRef = new ArrayList<T> ();
  maxLen = size-1;
 }
 public void push(T val) {
  if (stackRef.size() == maxLen)
   System.out.println(" Error en push - stack lleno");
  else
   stackRef.add(val);
...
```

Instanciación: Stack2<String> miStack = new Stack2<string> ();

```
    {
      stackPtr = new Type[size];
      maxLen = size - 1;
      topSub = -1;
    }
  ...
}
```

```
,
public void push(T val) {
  if (stackRef.size() == maxLen)
    System.out.println(" Error en push - stack lleno");
  else
    stackRef.add(val);
  ...
}
```

Orientación a Objetos

jueves, 21 de octubre de 2021 12:12

- **Conceptos Claves**
  - La ***programación Orientada a Objetos (POO)*** tiene tres aspectos:
    - Tipo de Dato Abstracto
    - Herencia
    - Polimorfismo
- **Sabores (wtf está en el pdf)**

- **Problemas de los *TDA* que ataca la *orientación a objetos***
  - Herencia
    - Permite extensión de datos u operaciones
    - Permite reutilizar el software que ya está presente
  - Redefinición
    - Permite redefinir un comportamiento, además de reutilizarlo
  - Abstracción
    - Se requiere abstraer operaciones similares para diferentes componentes en un nuevo tipo
  - Polimorfismo
    - Se requiere extender el tipo de datos sobre las cuales se pueden aplicar las operaciones
    - Existen diferentes tipos:
      - Sobrecarga
      - Tipos Parametrizados

- **Modelo Objetual**
  - Programa
    - Conjunto de objetos interactuantes
  - Clase
    - Corresponde a una declaración de tipo, que especifica el estado y comportamiento que tendrán sus instancias y objetos
    - El ***tipo*** queda definido por su ***interfaz***, que especifica métodos y constantes
    - La ***implementación*** del tipo queda especificado por las variables y código que define el comportamiento de los métodos y la manipulación de su estado
  - Objeto
    - Instancia concreta de una clase
  - Método
    - Especifica el comportamiento de los operadores de la clase, y controla el acceso al estado
    - Conjunto de métodos que define el protocolo para los mensajes
  - Mensaje
    - Invocación de un método que contiene identificador del objeto y método, y parámetros reales o resultados