Clase 8

Tipo de dato abstracto

- Abstraer es representar algo descubriendo sus características esenciales y suprimiendo las que no lo son
- El principio básico de la abstracción es la información oculta
- Los tipos de datos son abstracciones y el proceso de construir nuevos tipos se llama abstracción de datos.
- Los nuevos tipos de datos definidos por el usuario se llaman tipos abstractos de datos.

Los tipos de datos son abstracciones y el proceso de construir nuevos tipos se llama abstracción de datos. Estos nuevos tipos se llaman tipo abstractos de datos

Las TADs satisfacen:

- Encapsulamiento
  - La representación del tipo y las operaciones permitidas para el tipo se describen en una única unidad sintáctica
- Ocultamiento de información
  - La representación de los objetos y la implementación del tipo permanecen ocultos

Ejemplos de TADs

- Paquete en ADA
- Clase en C
- Clase en Java
- Modulo en Modula-2

Especificación de un TAD

- La especificación formal proporciona un conjunto de axiomas que describen el comportamiento de todas las operaciones.
- Ha de incluir una parte de sintaxis y una parte de semántica

  Por ejemplo:

  TAD nombre del tipo (valores que toma los datos del tipo)
  Sintaxis
  Operación(Tipo argumento, ...) -> Tipo resultado
  ..
  Semántica
  Operación(valores particulares argumentos) ⇒ expresión resultado

- Hay operaciones definidas por sí mismas que se consideran constructores del TAD
- Normalmente, se elige como constructor la operación que inicializa.

TAD: CLASES

- Una clase es un tipo definido por el usuario
- Una clase contiene la especificación de los datos que describen un objeto junto con la descripción de las acciones que un objeto conoce
- Agrega un segundo nivel de abstracción que consiste en agrupar las clases en jerarquías de clase. De forma que la clase hereda todas las propiedades de la superclase

Sistema de Tipos

- Reglas que permiten estructurar y organizar sus tipos
- El objetivo de estos es escribir programas seguros
- Conocer el sistema de tipos nos permite conocer de una mejor forma los aspectos semánticos del lenguaje
- Cuestiones a tener en cuenta al diseñar un lenguaje

Características:

- Provee mecanismos de expresión
  - Definir y asociar tipos
- Define reglas de resolución
  - Equivalencia de tipos {dos valores son del mismo tipo?}
  - Compatibilidad de tipos {puedo usar el tipo en este contexto?}
  - Interferencia de tipos {cual tipo se deduce del contexto?}
- Mientras más flexible el lenguaje, más complejo el sistema

Seguridad vs Flexibilidad

- Se dice que el sistema de tipos es fuerte cuando especifica restricciones sobre la forma en la que las operaciones involucran valores de diferentes tipos lo contrario es débil. Básicamente un sistema es fuerte cuando me define la mayor cantidad de restricciones (esto permite evitar errores de tipo lo antes posible, errores a futuro)
- Si no ofrece restricciones puedo hacer más cosas, por lo tanto, más errores a futuro.
- Se dice que es fuertemente tipado si todos los errores de tipo se detectan.

El compilador puede garantizar la ausencia de errores de tipo

```
a = 2
b= "2"
Concatenar (a,b) //retorna "22"
Sumar (a,b) //retorna 4
```

```
a = 2
b= "2"
Concatenar (a,b) //error de tipos
Sumar (a,b) //error de tipos
Concatenar (str(a),b) //retorna "22"
    Sumar (a,int(b)) //retorna 4
```

Python es Fuertemente Tipado y tiene tipado dimámico.

C es débilmente tipado y tiene tipado estático.

GOBSTONE Fuertemente Tipado y tipado estático.

Especificaciones

- Tipo y tiempo de chequeo
  - Tipos de ligadura
    - Estático {Ligadura en compilación}. Puede exigir lo siguiente
      - Se pueden utilizar tipos de dato predefinidos
      - Todas las variables se declaran con un tipo asociado
      - Todas las operaciones se especifican indicando los tipos de los operandos requeridos y el tipo resultado
    - Dinámico {ligaduras en ejecución}
      - Provoca más comprobaciones en tiempo de ejecución
      - Que sea dinámico no significa que sea mas inseguro.
    - Seguro {no es estático, ni inseguro}

- Reglas de equivalencia y conversión
  - Reglas semánticas si un tipo es valido en un contexto determinado
    - Equivalencia por nombre: Dos variables son del mismo tipo si están declaradas juntas o con el mismo nombre de tipo
    - Equivalencia por estructura: Dos variables son del mismo tipo si los componentes de su tipo son iguales

  - Dos tipos son compatibles si
    - Son equivalentes
    - Se pueden convertir

  - Coerción: significa convertir un valor de un tipo a otro.
    - **Widening** implica convertir un tipo de dato a otro más grande o más preciso, **narrowing** implica convertir un tipo de dato a otro más pequeño o menos preciso, y la cláusula de **casting** es una forma explícita de forzar la conversión de un tipo de dato a otro.

- Reglas de inferencia de tipos
  - La interferencia de tipos permite que el tipo de una entidad declarada se “deduzca” en lugar de ser declarado

• Un operador predefinido
fun f1(n,m)=(n mod m=0)

• Un operando
fun f2(n) = (n*2)

• Un argumento
fun f3(n:int) = n*n

• El tipo del resultado
fun f4(n):int = (n*n)

La inferencia puede realizarse de acuerdo al tipo

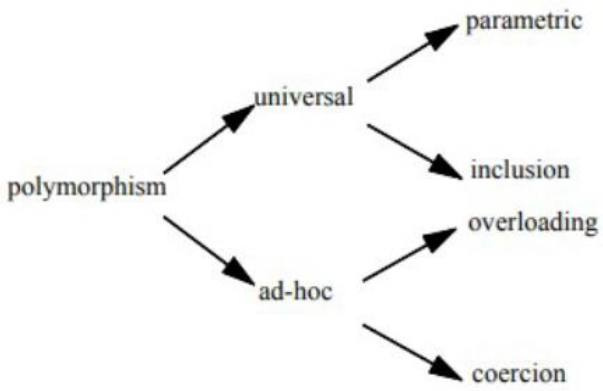
```
write(e,f(x)+1)

fun disjuntos(s1,s2:Conjunto):boolean;
```

• La función disjuntos deberá implementarse para cada tipo particular de conjunto?
• Las funciones read y write son “polimórficas”, pero no de forma pura (ya que el compilador infiere el tipo)

- Nivel de polimorfismo del lenguaje
  - Mono-mórfico
    - Un lenguaje se dice mono-mórfico si cada entidad se liga a un único tipo (estáticos)
    - En un lenguaje de programación mono-mórfico la función disjuntos deberá implementarse para cada tipo particular de conjunto
  - Polimórfico
    - Un lenguaje se dice polimórfico si las entidades pueden estar ligadas a más de un tipo
    - Las variables polimórficas pueden tomas valores de diferentes tipos
    - Las operaciones polimórficas son funciones que aceptan operandos de varios tipos
    - Los tipos polimórficos tienen operaciones polimórficas

Todos los lenguajes prácticos tienen cierto grado de polimorfismo, en consecuencia, las preguntas importantes a responder son

- ¿Qué diferentes tipos de polimorfismo se identifican?
- ¿Hasta dónde podemos llegar?



- El **polimorfismo universal** permite que una única operación se aplique uniformemente sobre un conjunto de tipos relacionados
    - Si la uniformidad de la estructura de tipos está dada a través de parámetros, hablamos de **polimorfismo paramétrico**
    - El **polimorfismo por inclusión** es otra forma de polimorfismo universal que permite modelar subtipos y herencia.
        - Si un tipo se define como un conjunto de valores y un conjunto de operaciones. Un subtipo T' de un tipo T puede definirse como un subconjunto de los valores de T y el mismo conjunto de operaciones.
        - El mecanismo de herencia permite definir una nueva clase derivada a partir de una clase base ya existente. Podría agregar atributos y comportamiento.

```
subtype TDíaDelMes is Integer range 1..31;
subtype TDíaFebrero is TDíaDelMes range 1..29;
subtype TLaborable is TDíaDeSemana range Lunes..Viernes;
```

- El **polimorfismo ad-hoc** permite que una función se aplique a distintos tipos con un comportamiento sustancialmente diferente en cada caso
    - El término **sobrecarga (overloading)** se utiliza para referirse a conjuntos de abstracciones diferentes que están ligadas al mismo símbolo o identificador.
    - La **coerción** permite que un operador que espera un operando de un determinado tipo T puede aplicarse de manera seguro sobre un operando de un tipo diferente al esperado