Estructuras de Datos

Tema 1. Programación Imperativa de Computadores

Tema 2. Fundamentos de Complejidad Algorítmica

Tema 3. Estructuras de datos jerárquicas

Tema 4. concepto y especificación de Tipos Abstractos de Datos (TADs)

Tema 5. Estructuras de datos lineales

Tema 6. Estructuras de datos jerárquicas

Bibliografía

Sahni, Sartaj, "Data structures, algorithms, and applications in Java". McGraw Hill, 2000

Weiss, Mark Allen, "Estructuras de datos y algoritmos". Addison-Wesley Iberoamericana, 1995.

Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, "Data structures and algorithms". Addison-Wesley, 1983.

Weiss, Mark Allen, "Estructuras de datos en Java : compatible con Java 2". Addison-Wesley, 2000

Michael T. Goodrich, Roberto Tamassia, "Data structures and algorithms in Java". John Wiley & Sons, 2006.

Objetivos

- Conocer el tipo abstracto de datos “Árbol”
- Saber usar el tipo abstracto de datos “Árbol”

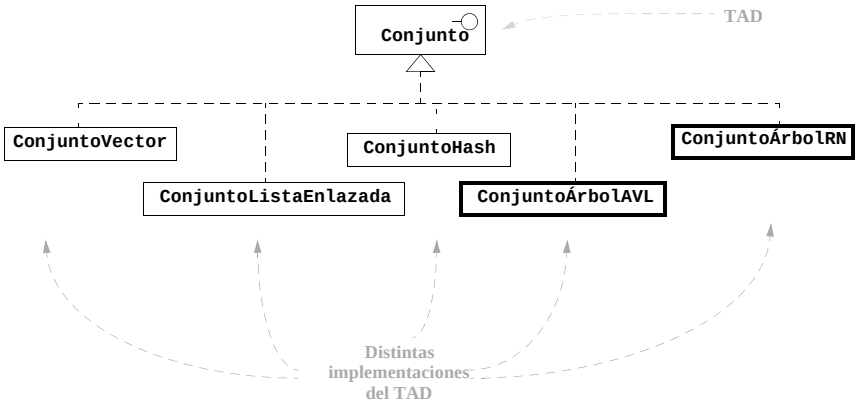
Tema 6. Estructuras de datos jerárquicas

1. *Introducción*
2. *El Tipo abstracto de datos árbol*
   - Operaciones habituales.
3. *Ejemplos con árboles*
   - Recorrido. Creación.

1 Introducción

Hemos utilizado árboles para implementar algunos TADs:

- **Bolsas, Conjuntos, Colas de Prioridad, Tablas**



Introducción (cont.)

Los árboles también constituyen un TAD **con numerosas aplicaciones directas:**

- **Representación de todo tipo de estructuras jerárquicas:**
  - **Organigrama de empresas**
  - **Redes de ordenadores (p.e. Internet)**
  - **Sistemas de ficheros**
  - **Sistemas de distribución (luz, agua, teléfono, ...)**
  - **...**
- **Árboles sintácticos para la representación y/o interpretación de términos de un lenguaje (de programación)**
- **Representación y/o interpretación de expresiones aritméticas**
- **Inteligencia Artificial y Juegos**
- **...**

2 El Tipo abstracto de datos árbol

Veremos un árbol genérico:

- **Número ilimitado de hijos en cada nodo**

Operaciones del árbol:

operación	argumentos	retorna	errores
constructor	Elemento	Árbol	
hazNulo			
estaVacio		booleano	
iterador		IteradorDeArbol	

El Iterador de árboles

La mayoría de las operaciones se encuentran en el iterador de árboles, que

- **contiene una referencia a uno de los nudos del árbol**
  - **inicialmente es la raíz**
  - **si la referencia es nula, se dice que el iterador *no es válido***
- **puede usarse para recorrer y/o modificar el árbol**
- **si el iterador no es válido, casi todas las operaciones lanzan `NoValido`**

Operaciones del iterador de árboles: operaciones de modificación

operación	argumentos	retorna	errores
constructor	elArbol	IteradorDeArbol	
insertaPrimerHijo	Elemento		NoValido
insertaSiguienteHermano	Elemento		EsRaiz, NoValido
eliminaHoja		Elemento	NoEsHoja, NoValido
modificaElemento	Elemento	Elemento viejo	NoValido
cortaRama		Rama cortada	NoValido
reemplazaRama	Nueva Rama	Rama cortada	NoValido
anadeRama	Nueva Rama		NoValido

Notas:

- *constructor*: Crea el iterador del árbol, con el nudo actual igual a la raíz, o no válido si el árbol está vacío
- *insertaPrimerHijo*: Añade un hijo al nudo actual, situado más a la izquierda que los actuales, y con el valor indicado
- *insertaSiguienteHermano*: Añade un hijo al padre del nudo actual, situándolo inmediatamente a la derecha del nudo actual. Lanza **`EsRaiz`** si se intenta añadir un hermano a la raíz
- *eliminaHoja*: Si el nudo actual es una hoja, la elimina del árbol y hace que el nudo actual sea su padre. Si no es una hoja, lanza **`NoEsHoja`**.
- *modificaElemento*: Modifica el contenido del nudo actual reemplazándolo por el **`elementoNuevo`**. Retorna el antiguo contenido del nudo actual
- *cortaRama*: Elimina la rama del árbol cuya raíz es en nudo actual, y hace que el nudo actual sea su padre. Retorna la rama cortada como un árbol independiente.
- *reemplazaRama*: reemplaza la rama del árbol cuya raíz es el nudo actual, sustituyéndola por **`nuevaRama`**; la posición actual no cambia, y será por tanto la raíz de **`nuevaRama`** en el árbol actual. Retorna la rama que ha sido reemplazada como un árbol independiente.
- *anadeRama*: Añade el árbol indicado por **`nuevaRama`** haciendo que su raíz sea hija del nudo actual, situándola a la derecha de los hijos actuales, si los hay

Operaciones del iterador de árboles: operaciones de consulta y recorrido

operación	argumentos	retorna	errores
irARaiz			
irAPrimerHijo			NoValido
irASiguienteHermano			NoValido
irAPadre			NoValido
contenido		Elemento	NoValido
esHoja		Booleano	NoValido
esRaiz		Booleano	NoValido
esUltimoHijo		Booleano	NoValido
esValido		Booleano	
clonar		IteradorDeArbol	

Notas:

- *contenido*: retorna el elemento contenido en el nudo actual
- *iaARaiz*: hace que el nudo actual sea la raíz del árbol; valdrá no válido si el árbol está vacío
- *irAPrimerHijo*: hace que el nudo actual sea el primer hijo del actual; valdrá no válido si el nudo actual no tiene hijos
- *irASiguienteHermano*: hace que el nudo actual sea el siguiente hermano del actual; valdrá no válido si el nudo actual no tiene hermanos derechos
- *irAPadre*: hace que el nudo actual sea el padre del actual; valdrá no válido si el nudo actual era la raíz
- *esHoja*: retorna un booleano que indica si el nudo actual es una hoja o no (es decir si no tiene hijos)
- *esRaiz*: retorna un booleano que indica si el nudo actual es la raíz del árbol
- *esUltimoHijo*: retorna un booleano que indica si el nudo actual es el último hijo de su padre (es decir si no tiene hermanos derechos)
- *esValido*: retorna un booleano que indica si el nudo actual es válido, o no
- *clonar*: retorna un iterador de árbol que es una copia del actual

3 Ejemplos con árboles

1. **Escribir métodos para mostrar el árbol en preorden, postorden e inorden**

2. **Escribir un programa para crear un árbol que represente una expresión, comenzando por la raíz y descendiendo a las hojas**

Ejemplo 1: mostrar en preorden

```
public static <E> void muestraEnPreorden
    (IteradorDeArbol<E> iterador)
{
   IteradorDeArbol<E> iter= iterador.clone();

   System.out.print(iter.contenido()+" ");

   iter.irAPrimerHijo();
   while (iter.esValido()) {
      muestraEnPreorden(iter);
      iter.irASiguienteHermano();
   }
}
```


Ejemplo 1: mostrar en postorden

```
public static <E> void muestraEnPostorden
   (IteradorDeArbol<E> iterador)
{
   IteradorDeArbol<E> iter= iterador.clone();

   E contenidoRaiz=iter.contenido();

   iter.irAPrimerHijo();
   while (iter.esValido()) {
      muestraEnPostorden(iter);
      iter.irASiguienteHermano();
   }

   System.out.print(contenidoRaiz+" ");
}
```

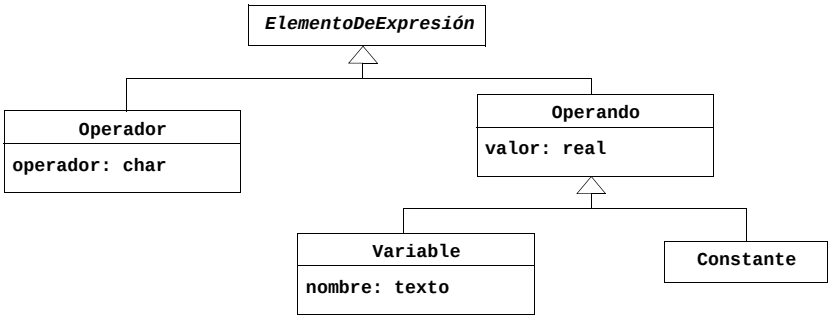
Ejemplo 1: mostrar en inorden

```
public static <E> void muestraEnInorden
   (IteradorDeArbol<E> iterador) {
   IteradorDeArbol<E> iter= iterador.clone();
   E contenidoRaiz=iter.contenido();
   if (iter.esHoja()) {
      System.out.print(contenidoRaiz);
   } else {
      System.out.print("(");
      iter.irAPrimerHijo();
      muestraEnInorden(iter);
      System.out.print(contenidoRaiz);
      iter.irASiguienteHermano();
      while (iter.esValido()) {
         muestraEnInorden(iter);
         iter.irASiguienteHermano();
      }
      System.out.print(")");
   }
}
```


Ejemplo 2: Creación del árbol

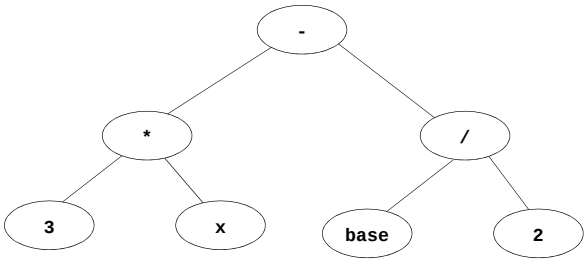
Usaremos las clases:

- **ElementoDeExpresion: superclase de todos los elementos**
- **Operador: representa un operador aritmético**
- **Variable: representa un operando variable**
- **Constante: representa un operando constante literal**



Ejemplo 2: Creación del árbol (cont.)

La expresión es: 3*x-base/2




```
// crear los operadores y operandos
Operador resta = new Operador('-');
Operador mult = new Operador('*');
Operador div =  new Operador('/');
Variable x =new Variable("x",1);
Variable base =new Variable("base",1);
Constante dos =new Constante(2);
Constante tres =new Constante(3);
```

Ejemplo 2: Creación del árbol (cont.)

```
Arbol<ElementoDeExpresion> arbol=
      new ArbolCE<ElementoDeExpresion>(resta);
IteradorDeArbol<ElementoDeExpresion> iter=
      arbol.iterador();

iter.insertaPrimerHijo(mult);
iter.irAPrimerHijo();  // mult
iter.insertaPrimerHijo(tres);
iter.irAPrimerHijo();  // tres
iter.insertaSiguienteHermano(x);
iter.irAPadre(); // mult
iter.insertaSiguienteHermano(div);
iter.irASiguienteHermano(); // div
iter.insertaPrimerHijo(base);
iter.irAPrimerHijo(); // base
iter.insertaSiguienteHermano(dos);
iter.irARaiz();
```


Ejemplo 2: Creación del árbol (cont.)

```
// mostrar el arbol en preorden
System.out.println("Arbol en preorden:");
OpArboles.muestraEnPreorden(iter);
System.out.println();
```