UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos 2006. **Trabajo práctico Nro. 7.**

Árboles Binarios

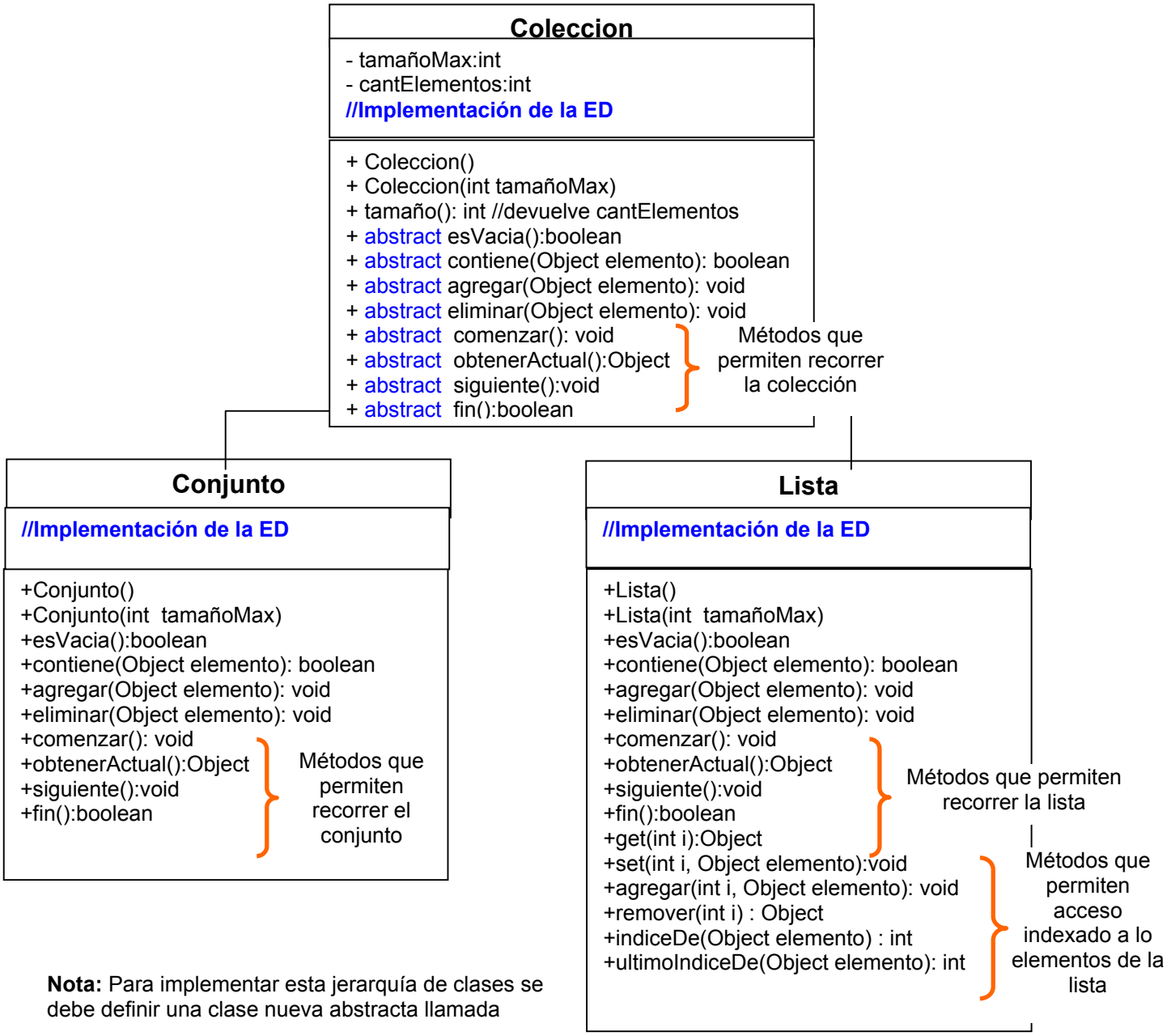
Objetivos

- Representar árboles binarios e implementar las operaciones de la abstracción
- Saber realizar distintos tipos de recorridos sobre árboles binarios
- Describir soluciones utilizando árboles binarios.

Ejercicio 1

Implementar la siguiente jerarquía de clases en el paquete **estructurasdedatos.colecciones**, de acuerdo a la siguiente especificación:

- La clase abstracta Coleccion representa un grupo de objetos llamados elementos. Algunas subclases de **Coleccion** permiten elementos duplicados, otras no: algunas son ordenadas y otras no. Esta clase modela un comporamiento común y general.
- La clase **Lista** y **Conjunto** son subclases concretas de **Coleccion** y mantienen la funcionalidad especificada en la practica 5.



Nota: Para implementar esta jerarquía de clases se debe definir una clase nueva abstracta llamada

Coleccion y adaptar las definiciones de **Lista** y **Conjunto** de la practica 5 para cumplir con la nueva funcionalidad.
Aclaración: El constructor sin argumentos debe dejar inicializada la variable tamañoMax en 100 y cantElementos en 0**.** El constructor que especifica un argumento debe asignar el valor del argumento a la variable tamañoMax y cantElementos en 0. Finalmente las subclases de colección que pueden crecer dinámicamente cuando llegan al tamaño máximo definido al momento de la instaciación, deben incrementar el valor de la variable tamañoMax en 1 y almacenar el nuevo elemento.

a) ¿A su criterio cuál cree que es el propósito de tener una clase abstracta como la definida anteriormente (la clase **Colección**)?
b) ¿Es posible obviar la implementación de un método abstracto definido en la superclase Colección en alguna de sus subclases?
c) Con respecto a los métodos que se utilizan para recorrer objetos Coleccion, ¿se podrían definir de otra manera alternativa a la definida anteriormente, de manera tal que puedan usarse en cualquier clase, inclusive en las no relacionadas con Coleccion mediante herencia, por ejemplo en ArbolBinario y obtener el mismo resultado? **Implemente la solución y haga los cambios necesarios en las clases definidas.**

Ejercicio 2
Un árbol binario con la representación de **hijo izquierdo e hijo derecho** puede implementarse

```
public class ArbolBinario {
        private Object dato;
        private ArbolBinario hijoIzquierdo, hijoDerecho;
}
```

Considere la siguiente especificación de la clase **ArbolBinario**

ArbolBinario
+ArbolBinario() +ArbolBinario(Object dato) +raiz():Object +hijoIzquierdo():ArbolBinario +hijoDerecho():ArbolBinario +agregarHijoIzquierdo(ArbolBinario unHijo) +agregarHijoDerecho(ArbolBinario unHijo) +eliminarHijoIzquierdo(ArbolBinario unHijo) +eliminarHijoDerecho(ArbolBinario unHijo)

a) Implementar la clase **ArbolBinario** en el paquete **estructuradedatos.arbolesbinarios**

Ejercicio 3
Agregue a la clase **ArbolBinario** los siguientes métodos y constructores:

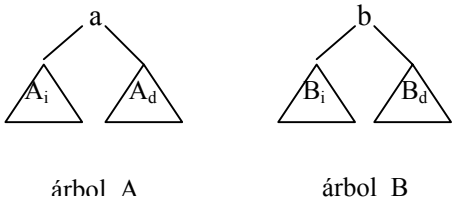
ArbolBinario
+ArbolBinario() +ArbolBinario(Object dato) +raiz():Object +hijoIzquierdo():ArbolBinario +hijoDerecho():ArbolBinario +agregarHijoIzquierdo(ArbolBinario unHijo) +agregarHijoDerecho(ArbolBinario unHijo) +eliminarHijoIzquierdo(ArbolBinario unHijo) +eliminarHijoDerecho(ArbolBinario unHijo) **+ArbolBinario (ArbolGeneral unArbol)** **+frontera(): Lista** **+es_menor(): boolean** **+zigZag(): boolean** **+lleno(): boolean** **+completo(): boolean** **+espejo(): ArbolBinario** **+colorear():void**

a) **ArbolBinario (ArbolGeneral unArbol).** Crea un arbol binario (bt) a partir de un árbol general (gt). La transformación es la siguiente. Cada vértice de *gt* estará representado por una hoja en *bt*. Si *gt* es una hoja entonces *bt* será una hoja con el mismo valor que el de *gt*. En otro caso, *gt* no es una hoja, entonces *bt* será armado como un vértice raíz, sin contenido, un subárbol izquierdo, *lt*, y un subárbol derecho, *rt*. *lt* es un árbol binario estricto formado a partir del subárbol más viejo de *gt* (el subárbol más izquierdo de *gt*) y *rt* es un árbol binario estricto, formado a partir de *gt* sin su subárbol más viejo.

b) **frontera(): Lista**. Se define **frontera** de un árbol binario, a las hojas de un árbol binario recorridos de izquierda a derecha. Escribir un método que dado un árbol binario devuelva su frontera. Nota: use la **clase Lista definida en el Trabajo Práctico Nro. 3.**

c) **esMenor(ArbolBinario unArbol): boolean.** Se define una relación de orden entre árboles binarios no nulos de enteros de la siguiente forma:

$$A < B \begin{cases} a < b; \\ a = b \text{ y } Ai < Bi \\ a = b \text{ y } Ai = Bi \quad \text{y } Ad < Bd \end{cases}$$

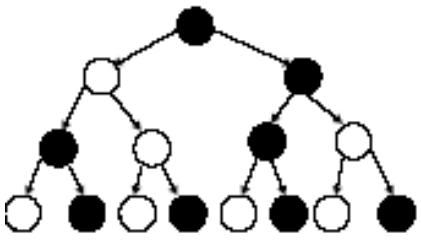


donde a, b son las raíces y Ai, Ad, Bi, Bd son los subárboles izquierdos y derechos.

d) **zigZag(): boolean.** Devuelve true si el árbol es degenerado con direcciones alternadas, esto es, si en lugar de ser una lista donde todos los hijos están en el lado izquierdo o derecho se van alternando.
e) **lleno(): boolean.** Devuelve true si el árbol es lleno, false en otro caso. Un árbol binario es lleno si tiene todas las hojas al mismo nivel y además tiene todas las hojas posibles, es decir todos los nodos intermedios tienen dos hijos.
f) **completo(): boolean.** Devuelve true si el árbol es completo, false en otro caso. Un árbol binario de altura h es completo si es lleno hasta el nivel h-1 y el nivel h se completa de izquierda a derecha.

g) **espejo(): ArbolBinario.** Por ejemplo:

h) **colorear().:void.** Un árbol binario coloreado es un árbol binario lleno cuyos nodos tienen dos colores posibles negro o blanco. El color para la raíz del árbol es negro. Y para cada nivel los colores de los nodos se intercalan. Como así también al comenzar cada nivel. Por ejemplo un árbol coloreado de altura 3 deberá ser:



Ejercicio 4
Resolver los siguientes problemas en JAVA

a) Se quiere hacer que una computadora adivine un nombre de animal que un usuario ha elegido previamente. La computadora debe hacer preguntas a las que el usuario debe responder por si o no. Con estas respuestas la computadora llega a determinar un animal, y arriesga una respuesta (haciendo una pregunta del tipo *Es un perro*?). Si acierta, el juego termina y la computadora adivinó. En otro caso, el usuario debe proveer una nueva pregunta que caracterice al animal en cuestión, la que debe ser incorporada a las ya existentes en el lugar que se considere adecuado.
a.1.) Defina una clase llamada Juego e implemente el método de instancia jugar():void.
a.2) Defina una clase TestJuego con un método main que se encargue de construir un árbol con al menos 3 niveles y luego comience el juego.

b) Considere un árbol binario no vacío con dos tipos de nodos, nodos **MIN** y nodos **MAX**. Cada nodo tiene un valor entero asociado. Se puede definir el valor de un árbol de estas características de la siguiente manera:

Si la raíz es un nodo **MIN**, entonces el valor del árbol es igual al mínimo valor entre:

- El entero almacenado en la raíz.
- El valor correspondiente al subárbol izquierdo, si el mismo no es vacío.
- El valor correspondiente al subárbol derecho, si el mismo no es vacío.

Si la raíz es un nodo **MAX**, entonces el valor del árbol es igual al máximo valor entre los valores nombrados anteriormente.

b.1.) Defina una clase ArbolMinMax como subclase de la clase ArbolBinario y defina los métodos de instancia setNodoMin():void, setNodoMax():void y evaluar(): int.

b.2) Defina una clase TestMiniMax que su método main se encargue de definir un arbol con al menos 7 nodos con diferentes valores y naturaleza (min o max) y lo evalúe.