UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Redictado 2017

Práctica 5
Árboles Binarios de Búsqueda

Importante: Puede continuar trabajando en su proyecto AyED. El archivo zip descargado desde la página de la cátedra no es un proyecto eclipse, por lo tanto:

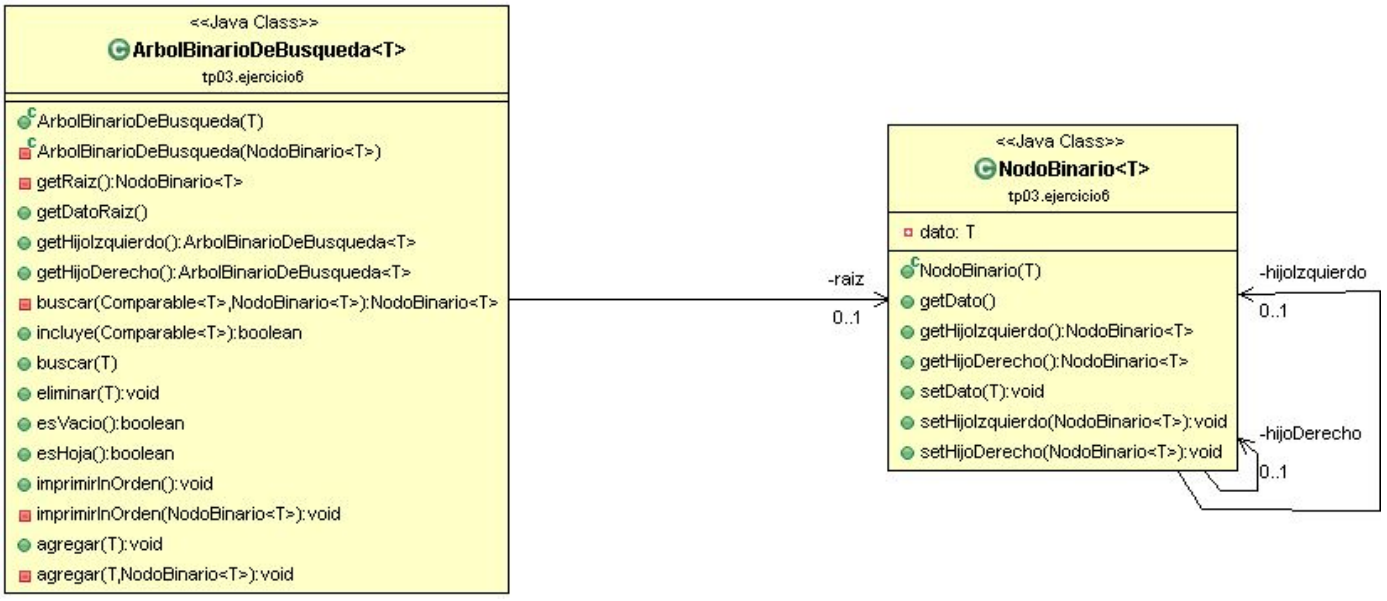
1. Descomprima el archivo zip.
2. Sobre la carpeta **src** de su proyecto AyED haga click con el botón derecho del mouse y seleccione la opción *Import* > *FileSystem*.
3. Haga click en "Browse", busque la carpeta descomprimida y seleccione la carpeta **src** (haga click para que aparezca el check seleccionado).
4. Haga click en el botón finalizar.

Objetivos

- Representar árboles binarios de búsqueda e implementar las operaciones de la abstracción
- Realizar distintos tipos de recorridos sobre árboles binarios de búsqueda
- Describir soluciones utilizando árboles binarios de búsqueda

Ejercicio 1

Considere la siguiente especificación de la clase **ArbolBinarioDeBusqueda** (con la representación hijo izquierdo e hijo derecho):



Aclaración:
En la imagen no aparece la definición de la variable de tipo en la clase ArbolBinarioDeBusqueda. La misma es **<T extends Comparable<T>>**, lo cual indica que la clase representada por la variable T implemente la interface Comparable.
Las clases que implementan la interface *java.lang.Comparable<T>* permiten que sus instancias se puedan comparar entre sí. Para lograr esto, deben implementar el método compareTo(T), el cual retorna el resultado de comparar el receptor del mensaje con el parámetro recibido. Este valor se codifica con un entero, el cual presenta la siguiente característica:

= 0: si el objeto receptor es igual al pasado en el argumento.
> 0: si el objeto receptor es mayor que el pasado como parámetro.
< 0: si el objeto receptor es menor que el pasado como parámetro.

La descripción de cada método es la siguiente:

El constructor **ArbolBinarioDeBusqueda()** inicializa un árbol binario de búsqueda vacío con la raíz en null.

El constructor **ArbolBinarioDeBusqueda(T dato)** inicializa un árbol que tiene como raíz un nodo binario de búsqueda. Este nodo tiene el dato pasado como parámetro y ambos hijos nulos.

El constructor **ArbolBinarioDeBusqueda(NodoBinario<T> nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. Este método es privado y se podrá usar en la implementación de las operaciones sobre el árbol.

El método **getRaiz():NodoBinario <T>** retorna el nodo ubicado en la raíz del árbol.

El método **getDatoRaiz():T** retorna el dato almacenado en el NodoBinario raíz del árbol, sólo si el árbol no es vacío.

El método **esVacio(): boolean** indica si el árbol es vacío (no tiene dato cargado).

Los métodos **getHijoIzquierdo():ArbolBinarioDeBusqueda<T>** y **getHijoDerecho():ArbolBinarioDeBusqueda<T>** retornan los árboles hijos que se ubican a la izquierda y derecha del nodo raíz respectivamente. Están indefinidos para un árbol vacío.

El método **incluye (T dato)** retorna un valor booleano indicando si el dato recibido está incluido en el árbol.

El método **buscar (T dato):T** retorna el valor almacenado en el árbol que es igual al dato recibido.

El método **agregar (T dato)** agrega el dato indicado al árbol. En caso de encontrar un elemento igual dentro del árbol, reemplaza el existente por el recibido.

El método **eliminar (T dato)** elimina el dato del árbol.

> **Nota:** Tener presente que en ABB no se pueden almacenar cualquier objeto, ya que estos necesitan ser comparables para poder ordenarlos dentro de la estructura.

a) Analice la implementación en JAVA de la clase **ArbolBinarioDeBusqueda** brindada por la cátedra y responda las siguientes preguntas:

- ¿Qué métodos de la clase **ArbolBinarioDeBusqueda** son iguales a los de la clase **ArbolBinario**? Justifique.
- ¿Qué métodos de la clase **ArbolBinarioDeBusqueda** son diferentes a los de la clase **ArbolBinario**? Justifique.

Ejercicio 2
Implementar dentro de la clase ABB el método **datosMayores(T dato): ListaGenerica<T>**, que devuelve una lista con todos los datos mayores a un dato dado.

Ejercicio 3
Implementar dentro de la clase ABB el método **imprimirDatosOrdenados()**, que imprime por consola los datos del árbol en forma ordenada.

Ejercicio 4
Definir la clase **SiuGuarani** que tenga una variable de instancia que sea un **ArbolBinarioDeBusqueda** de tipo **Alumno**. El ABB estará ordenado por el número de legajo de los alumnos. Luego, implementar el método **getDatosAlumno(Alumno alumno):String**, que dado un alumno, lo busque en el árbol y retorne un String con toda la información del mismo.

```
public class SiuGuarani  {
  private ArbolBinarioDeBusqueda<Alumno> datos;
  //otros métodos...
  public class String getDatosAlumno(Alumno alumno) {
        //código
  }
}
```

La clase **Alumno**, debe tener la siguiente información: legajo, nombre, apellido, año que ingresó, y localidad.

Ejercicio 5
Analizar cómo quedaría un Árbol Binario de Búsqueda si se permitieran claves repetidas. ¿Qué métodos se modificarían? Definir una política y plantear un pseudocódigo de los métodos que mencionó.