**TAREA DE ESTRUCTURAS DE DATOS**

**ÁRBOLES DE BÚSQUEDA BINARIA**

Realizado por Byron Alvarez

(archivos JAVA adjuntos en la asignación y en el documento en modo de texto)

**1. Enunciado**: Diseñe un programa en JAVA, que permita el almacenamiento de números extraídos desde una matriz **A** en un árbol binario de búsqueda. Deberá cumplirse los siguientes requerimientos:

 Deberá existir un método que permita el ingreso de números en una matriz de tamaño **n** x **n** desde teclado (**solamente bajo la diagonal principal**).

 Los números extraídos desde la matriz deberán almacenarse según su valor en el árbol binario.

 Deberá almacenarse además en cada nodo el número de veces que se repite cada número.

 Permitir conocer en orden, los números ingresados y cuantas veces se repiten.

Código Utilizado para la resolución del enunciado

|  |
| --- |
| package enunciado01;  import java.util.Scanner;  /\*  Clase Nodo la cual nos sirve para cada nodo definido del arbol binario  \*/  class Nodo {  int valor;  int repeticiones;  Nodo izquierdo, derecho;  public Nodo(int valor) {  this.valor = valor;  this.repeticiones = 1;  this.izquierdo = this.derecho = null;  }  }  /\*  Clase necesaria para implementar las funciones del arbol binario como la inserción  \*/  class ArbolBinarioBusqueda {  Nodo raiz;  public ArbolBinarioBusqueda() {  raiz = null;  }  /\*  Funcion que inserta un nuevo nodo desde el main, esta funcion llama a una  funcion recursiva la cual trabaja desde la clase  \*/  public void insertar(int valor) {  raiz = insertarRec(raiz, valor);  }  /\*  Funcion insertar recursiva, ingresa nuevos nodos comparando su valor con la  raiz (si es mayor al nodo va al nodo derecho, sino al izquierdo) y si se  repite aumenta la varible contador definida  \*/  private Nodo insertarRec(Nodo raiz, int valor) {  if (raiz == null) {  raiz = new Nodo(valor);  return raiz;  }  if (valor < raiz.valor) {  raiz.izquierdo = insertarRec(raiz.izquierdo, valor);  } else if (valor > raiz.valor) {  raiz.derecho = insertarRec(raiz.derecho, valor);  } else {  // Si el valor ya existe, incrementa el contador de repeticiones  raiz.repeticiones++;  }  return raiz;  }  public void mostrarEnOrden() {  mostrarEnOrdenRec(raiz);  }  /\*  Muestra los valores del arbol binario, muestra por pantalla el numero y  la cantidad de repeticiones del mismo  \*/  private void mostrarEnOrdenRec(Nodo raiz) {  if (raiz != null) {  mostrarEnOrdenRec(raiz.izquierdo);  System.out.println("Número: " + raiz.valor + ", Repeticiones: " + raiz.repeticiones);  mostrarEnOrdenRec(raiz.derecho);  }  }  }  public class Enunciado01 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.print("Ingrese el tamaño de la matriz (n): ");  int n = sc.nextInt();  int[][] matriz = new int[n][n];  System.out.println("Ingrese los números bajo la diagonal principal:");  for (int i = 0; i < n; i++) {  for (int j = 0; j <= i; j++) {  System.out.print("Elemento [" + i + "][" + j + "]: ");  matriz[i][j] = sc.nextInt();  }  }  ArbolBinarioBusqueda abb = new ArbolBinarioBusqueda();  // Insertar los elementos de la matriz en el árbol binario  for (int i = 0; i < n; i++) {  for (int j = 0; j <= i; j++) {  abb.insertar(matriz[i][j]);  }  }  // Mostrar los números en orden y sus repeticiones  System.out.println("Números en orden y sus repeticiones:");  abb.mostrarEnOrden();  }  } |

**2. Enunciado:** Realice un programa en Java, que permita almacenar en ABB los siguientes datos de Empleados:

|  |  |  |
| --- | --- | --- |
| a. | **Nombre** | // leído desde teclado |
| b.  c. | **Id**  **Categoría (A, B o C)** | // leído desde teclado  // leído desde teclado |
| d. | **Antigüedad** | // leído desde teclado |
| e. | **Sueldo total** | // Calculado de acuerdo a los siguientes parámetros |

 Categoría A: Bonificaciones: de $800 por Coordinación y $500 por

responsabilidad más $30 por cada año de antigüedad a partir del 5to año

 Categoría B: Bonificaciones: de $400 por Coordinación y de $300 por responsabilidad más $30 por cada año de antigüedad a partir del 5to año

 Categoría C: Bonificaciones: de $300 por responsabilidad más $25 por cada año de antigüedad a partir del 5to año

Sueldo básico $450

Descuentos: Si el total es mayor a $2100, se aplicará el 8% de Impuesto a la

Renta, caso contrario 5%

El programa deberá almacenar los datos de los empleados en árboles binarios de acuerdo a su categoría (**A**, **B** y **C**) y ordenados por su **Id**.

Se requiere la presentación ordenada (de acuerdo a la categoría) de: **Id**, **nombre** y

**sueldo total a recibir**.

Código utilizado para la resolución del enunciado

|  |
| --- |
| package enunciado02;  import java.util.Scanner;  /\*  Clase Empleado la cual nos ayuda a definir y almacenar la informacion de cada  empleado a ingresar  \*/  class Empleado {  String nombre;  int id;  char categoria;  int antiguedad;  double sueldoTotal;  /\*  Constructor definido para asignar valores a un objeto de tipo Empleado  \*/  public Empleado(String nombre, int id, char categoria, int antiguedad) {  this.nombre = nombre;  this.id = id;  this.categoria = categoria;  this.antiguedad = antiguedad;  this.sueldoTotal = calcularSueldoTotal();  }  /\*  Funcion que calcula los sueldos totales de cada empleado según la condición  especificada (Categoría A, B o C)  \*/  private double calcularSueldoTotal() {  double sueldoBasico = 450;  double bonificacion = 0;  switch (categoria) {  case 'A':  bonificacion = 800 + 500;  if (antiguedad > 5) {  bonificacion += 30 \* (antiguedad - 5);  }  break;  case 'B':  bonificacion = 400 + 300;  if (antiguedad > 5) {  bonificacion += 30 \* (antiguedad - 5);  }  break;  case 'C':  bonificacion = 300;  if (antiguedad > 5) {  bonificacion += 25 \* (antiguedad - 5);  }  break;  }  double sueldoTotal = sueldoBasico + bonificacion;  // Aplicar descuentos  if (sueldoTotal > 2100) {  sueldoTotal \*= 0.92; // 8% de impuesto a la renta  } else {  sueldoTotal \*= 0.95; // 5% de impuesto a la renta  }  return sueldoTotal;  }  }  /\*  Clase Nodo la cual nos sirve para cada nodo definido del arbol binario  \*/  class Nodo {  Empleado empleado;  Nodo izquierdo, derecho;  public Nodo(Empleado empleado) {  this.empleado = empleado;  this.izquierdo = this.derecho = null;  }  }  /\*  Clase necesaria para implementar las funciones del arbol binario como la inserción  y mmostrar informacion en orden  \*/  class ArbolBinarioBusqueda {  Nodo raiz;  public ArbolBinarioBusqueda() {  raiz = null;  }  public void insertar(Empleado empleado) {  raiz = insertarRec(raiz, empleado);  }  /\*  Metodo de insercion recursiva utilizado de manera similar al Enunciado01  \*/  private Nodo insertarRec(Nodo raiz, Empleado empleado) {  if (raiz == null) {  raiz = new Nodo(empleado);  return raiz;  }  if (empleado.id < raiz.empleado.id) {  raiz.izquierdo = insertarRec(raiz.izquierdo, empleado);  } else if (empleado.id > raiz.empleado.id) {  raiz.derecho = insertarRec(raiz.derecho, empleado);  }  return raiz;  }  public void mostrarEnOrden() {  mostrarEnOrdenRec(raiz);  }  /\*  Muestra la información de los empleados almacenados en el arbol binario  \*/  private void mostrarEnOrdenRec(Nodo raiz) {  if (raiz != null) {  mostrarEnOrdenRec(raiz.izquierdo);  System.out.println("ID: " + raiz.empleado.id + ", Nombre: " + raiz.empleado.nombre + ", Sueldo Total: " + raiz.empleado.sueldoTotal);  mostrarEnOrdenRec(raiz.derecho);  }  }  }  public class Enunciado02 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  /\*  Creacion de un arbol por cada categoria  \*/  ArbolBinarioBusqueda arbolA = new ArbolBinarioBusqueda();  ArbolBinarioBusqueda arbolB = new ArbolBinarioBusqueda();  ArbolBinarioBusqueda arbolC = new ArbolBinarioBusqueda();  System.out.print("Ingrese la cantidad de empleados: ");  int cantidadEmpleados = sc.nextInt();  sc.nextLine(); // Limpieza de Buffer  for (int i = 0; i < cantidadEmpleados; i++) {  System.out.println("\nEmpleado #" + (i+1));  System.out.print("Nombre: ");  String nombre = sc.nextLine();  System.out.print("ID: ");  int id = sc.nextInt();  System.out.print("Categoría (A, B o C): ");  char categoria = sc.next().charAt(0);  System.out.print("Antigüedad: ");  int antiguedad = sc.nextInt();  sc.nextLine(); // Consumir el salto de línea  Empleado empleado = new Empleado(nombre, id, categoria, antiguedad);  /\*  Ingreso del empleado al arbol binario segun la categoria ingresada  por pantalla  \*/  switch (categoria) {  case 'A':  arbolA.insertar(empleado);  break;  case 'B':  arbolB.insertar(empleado);  break;  case 'C':  arbolC.insertar(empleado);  break;  }  }  System.out.println("\nEmpleados en categoría A:");  arbolA.mostrarEnOrden();  System.out.println("\nEmpleados en categoría B:");  arbolB.mostrarEnOrden();  System.out.println("\nEmpleados en categoría C:");  arbolC.mostrarEnOrden();  }  } |

**3. Enunciado:** Realice un programa en Java, que permita almacenar en ABB los siguientes datos de Clientes:

a. **Nombre** // leído desde teclado b. **Id** // leído desde teclado c. **Antigüedad** // leído desde teclado d. **Historial crediticio (A, B, C)** // leído desde teclado

e. **Categoría** // Calculado de acuerdo a los siguientes parámetros

i. Monto de compra > $1000  3 puntos; entre 500 y 1000  2 puntos; menor a 500  0 puntos

ii. Historial crediticio: A  3 puntos; B  2 puntos y C  1 punto.

iii. Antigüedad: > a 5 años  3 puntos; entre 2 y 5 años  2 puntos y menor a 2 1 punto

Categoría A: 7 puntos o más Categoría B: entre 3 y 7 puntos Categoría C: menor a 3 puntos

Descuentos: Categorías (A 10%, B 6% y C 2%)

El programa deberá almacenar los datos de los Clientes en árboles binarios de acuerdo a su categoría (**A**, **B** y **C**) y ordenados por su **Id**.

Se requiere la presentación ordenada (de acuerdo a la categoría) de: **Id**, **nombre** y

**descuento a aplicar**.

Código utilizado para la resolución del enunciado

|  |
| --- |
| package enunciado03;  import java.util.Scanner;  /\*  Clase Cliente la cual nos ayuda a definir y almacenar la informacion de cada  empleado a ingresar  \*/  class Cliente {  String nombre;  int id;  int antiguedad;  char historialCrediticio;  char categoria;  double descuento;  double montoCompra;  /\*  Constructor definido para asignar valores a cualquier instancia de tipo Cliente  \*/  public Cliente(String nombre, int id, int antiguedad, char historialCrediticio, double montoCompra) {  this.nombre = nombre;  this.id = id;  this.antiguedad = antiguedad;  this.historialCrediticio = historialCrediticio;  this.montoCompra = montoCompra;  this.categoria = calcularCategoria(montoCompra);  this.descuento = calcularDescuento();  }  /\*  Función que calcula la categoría en base a los parametros designados  \*/  private char calcularCategoria(double montoCompra) {  int puntos = 0;  // Calcular puntos por monto de compra  if (montoCompra > 1000) {  puntos += 3;  } else if (montoCompra >= 500) {  puntos += 2;  }  // Calcular puntos por historial crediticio  switch (historialCrediticio) {  case 'A':  puntos += 3;  break;  case 'B':  puntos += 2;  break;  case 'C':  puntos += 1;  break;  }  // Calcular puntos por antiguedad  if (antiguedad > 5) {  puntos += 3;  } else if (antiguedad >= 2) {  puntos += 2;  } else {  puntos += 1;  }  // Determinar categoría  if (puntos >= 7) {  return 'A';  } else if (puntos >= 3) {  return 'B';  } else {  return 'C';  }  }  /\*  Una vez designada la categoria con la función anterior, ocupamos la funcion  para calcular el descuento en base a la categoria a la que pertenece el cliente  \*/  private double calcularDescuento() {  switch (categoria) {  case 'A':  return 0.10;  case 'B':  return 0.06;  case 'C':  return 0.02;  default:  return 0.0;  }  }  }  /\*  Clases Nodo Y ArbolBinario utilizadas en un contexto similar al las funciones  del Enunciado02  \*/  class Nodo {  Cliente cliente;  Nodo izquierdo, derecho;  public Nodo(Cliente cliente) {  this.cliente = cliente;  this.izquierdo = this.derecho = null;  }  }  class ArbolBinarioBusqueda {  Nodo raiz;  public ArbolBinarioBusqueda() {  raiz = null;  }  public void insertar(Cliente cliente) {  raiz = insertarRec(raiz, cliente);  }  private Nodo insertarRec(Nodo raiz, Cliente cliente) {  if (raiz == null) {  raiz = new Nodo(cliente);  return raiz;  }  if (cliente.id < raiz.cliente.id) {  raiz.izquierdo = insertarRec(raiz.izquierdo, cliente);  } else if (cliente.id > raiz.cliente.id) {  raiz.derecho = insertarRec(raiz.derecho, cliente);  }  return raiz;  }  public void mostrarEnOrden() {  mostrarEnOrdenRec(raiz);  }  private void mostrarEnOrdenRec(Nodo raiz) {  if (raiz != null) {  mostrarEnOrdenRec(raiz.izquierdo);  System.out.println("ID: " + raiz.cliente.id + ", Nombre: " +  raiz.cliente.nombre + ", Descuento: " +  (raiz.cliente.descuento \* 100) + "%" + " ; Monto de compra: " + raiz.cliente.montoCompra);  mostrarEnOrdenRec(raiz.derecho);  }  }  }  public class Enunciado03 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  /\*  Creacion de un arbol binario por categoria para la asignación de los  nodos (clientes)  \*/  ArbolBinarioBusqueda arbolA = new ArbolBinarioBusqueda();  ArbolBinarioBusqueda arbolB = new ArbolBinarioBusqueda();  ArbolBinarioBusqueda arbolC = new ArbolBinarioBusqueda();  System.out.print("Ingrese la cantidad de clientes: ");  int cantidadClientes = sc.nextInt();  sc.nextLine(); // Limpieza de Buffer  for (int i = 0; i < cantidadClientes; i++) {  System.out.println("\nCliente #" + (i+1));  System.out.print("Nombre: ");  String nombre = sc.nextLine();  System.out.print("ID: ");  int id = sc.nextInt();  System.out.print("Antigüedad: ");  int antiguedad = sc.nextInt();  System.out.print("Historial crediticio (A, B o C): ");  char historialCrediticio = sc.next().charAt(0);  System.out.print("Monto de compra: ");  double montoCompra = sc.nextDouble();  sc.nextLine(); // Consumir el salto de línea  Cliente cliente = new Cliente(nombre, id, antiguedad, historialCrediticio, montoCompra);  switch (cliente.categoria) {  case 'A':  arbolA.insertar(cliente);  break;  case 'B':  arbolB.insertar(cliente);  break;  case 'C':  arbolC.insertar(cliente);  break;  }  }  System.out.println("\nClientes en categoría A:");  arbolA.mostrarEnOrden();  System.out.println("\nClientes en categoría B:");  arbolB.mostrarEnOrden();  System.out.println("\nClientes en categoría C:");  arbolC.mostrarEnOrden();  }  } |