<Estructuras definidas para testers>

Tabla de contenido

LISTA 1:

```
Elementos lista 1:
1 | 5 | 6 | 8 | 14 | 15 | 20 | 22 |
Elementos lista 2:
4 | 7 | 8 | 9 | 20 | 21 |
```

```
/*Dos listas de enteros y una tercera vacía*/
PositionList<Integer> L1 = new ListaDoblementeEnlazada<Integer>();
            L1.addLast(1); L1.addLast(5); L1.addLast(6); L1.addLast(8); L1.addLast(14);
L1.addLast(15); L1.addLast(20); L1.addLast(22);


            Iterator<Integer> it1 = L1.iterator();


            System.out.println("Elementos lista 1: ");
            while (it1.hasNext())   //Mientras el iterador tenga siguiente.
                  System.out.print(it1.next() + " | "); //Imprime el elemento siguiente.
            System.out.println();


            PositionList<Integer> L2 = new ListaDoblementeEnlazada<Integer>();
            L2.addLast(4); L2.addLast(7); L2.addLast(8); L2.addLast(9); L2.addLast(20);
L2.addLast(21);


            Iterator<Integer> it2 = L2.iterator();


            System.out.println("Elementos lista 2: ");
            while (it2.hasNext())   //Mientras el iterador tenga siguiente.
                  System.out.print(it2.next() + " | "); //Imprime el elemento siguiente.
            System.out.println();


            PositionList<Integer> L3 = new ListaDoblementeEnlazada<Integer>();
            Iterator<Integer> it = L3.iterator();


            System.out.println("Elementos lista 3: ");
            while (it.hasNext())    //Mientras el iterador tenga siguiente.
                  System.out.print(it.next() + " | "); //Imprime el elemento siguiente.
            System.out.println();
```

LISTA 2:

```
Cadena: { d e e x e e d e e d }
```

```
/*Lista de caracteres*/

PositionList<Character> lista = new ListaDoblementeEnlazada<Character>(); //deexeedeed
            lista.addLast('d');lista.addLast('e'); lista.addLast('e');
            lista.addLast('x');

            lista.addLast('e'); lista.addLast('e'); lista.addLast('d');
            lista.addLast('e'); lista.addLast('e'); lista.addLast('d');

            System.out.print("Cadena: { ");
            for(Character charac : lista)
                  System.out.print(charac + " ");
            System.out.println("}");
```

MAPEO:

```
* Mapeo 1:
(0,345) | (2,85) | (4,75) | (6,53) | (8,26) | (10,66) |
* Mapeo 2:
(1,63) | (2,85) | (3,46) | (4,75) | (5,88) | (6,52) | (7,78) | (8,26) | (9,63) | (10,34) |
```

```
/*Dos mapeos de enteros con enteros, con algunas entradas repetidas entre si*/
Map<Integer,Integer> m1 = new MapeoConLista<Integer, Integer>();
            Map<Integer,Integer> m2 = new MapeoConLista<Integer, Integer>();

            try {

                  m1.put(0,  345);
                  m1.put(2, 85);
                  m1.put(4, 75);
                  m1.put(6, 53);
                  m1.put(8, 26);
                  m1.put(10, 66);

                  m2.put(1, 63);
                  m2.put(2, 85);
                  m2.put(3, 46);
                  m2.put(4, 75);
                  m2.put(5, 88);
                  m2.put(6, 52);
                  m2.put(7, 78);
                  m2.put(8, 26);
                  m2.put(9, 63);
                  m2.put(10, 34);

                  PositionList<Entry<Integer,Integer>> l = entradasRepetidas(m1,m2);

                  System.out.println("* Mapeo 1: ");
                  for(Entry<Integer,Integer> e : m1.entries())
```

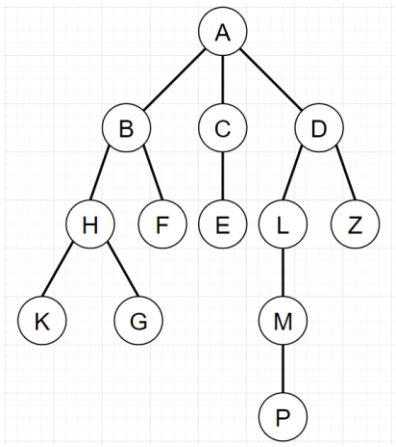
```
                  System.out.print("(" + e.getKey() + "," + e.getValue() + ")"  + "
| ");


            System.out.println();
            System.out.println("* Mapeo 2: ");
            for(Entry<Integer,Integer> e : m2.entries())
                  System.out.print("(" + e.getKey() + "," + e.getValue() + ")"  + "
| ");
```

DICCIONARIO:

```
* Diccionario:
(a,15) | (a,98) | (b,20) | (c,6) | (c,788) | (c,1000) | (d,1500) | (e,26) | (e,42) | (f,400) | (f,88) |
```

```
/*Diccionario de claves de caracteres y valores numéricos*/

Dictionary<Character,Integer> dicc = new DiccionarioHashAbierto<Character,Integer>();


		try {


			dicc.insert('a', 15);
			dicc.insert('b', 20);
			dicc.insert('f', 400);
			dicc.insert('c', 6);
			dicc.insert('a', 98);
			dicc.insert('e', 26);
			dicc.insert('c', 788);
			dicc.insert('c', 1000);
			dicc.insert('d', 1500);
			dicc.insert('e', 42);
			dicc.insert('f', 88);


			/*Imprimir diccionario*/
			System.out.println("* Diccionario: ");
			for(Entry<Character,Integer> e : dicc.entries())
				System.out.print(e.toString() + " | ");
			System.out.println("\n");


		} catch (InvalidKeyException e1) {
			e1.getMessage();
		}
```

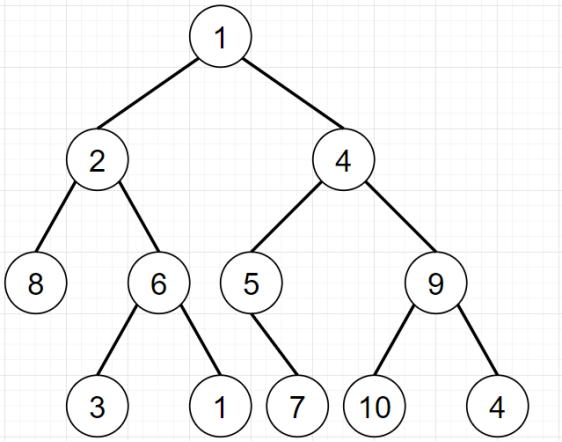
ÁRBOL:




```
/*Árbol general de caracteres*/
Arbol<Character> MyTree = new Arbol<Character>(); //Creo el arbol vacio.

            try {

                MyTree.createRoot('A');
                Position<Character> nodoB = MyTree.addFirstChild(MyTree.root(), 'B');
//Hijos de A (raiz).
                Position<Character> nodoC = MyTree.addLastChild(MyTree.root(), 'C');
              Position<Character> nodoD = MyTree.addLastChild(MyTree.root(), 'D');

              Position<Character> nodoH = MyTree.addFirstChild(nodoB, 'H'); //Hijos de B
              Position<Character> nodoF = MyTree.addLastChild(nodoB, 'F');

              Position<Character> nodoK = MyTree.addFirstChild(nodoH, 'K'); //Hijos de H
              Position<Character> nodoG = MyTree.addLastChild(nodoH, 'G');

              Position<Character> nodoE = MyTree.addFirstChild(nodoC, 'E'); //Hijo de C

              Position<Character> nodoL = MyTree.addFirstChild(nodoD, 'L'); //Hijos de D
              Position<Character> nodoZ = MyTree.addLastChild(nodoD, 'Z');

              Position<Character> nodoM = MyTree.addFirstChild(nodoL, 'M'); //Hijo de L
```

```
            Position<Character> nodoP = MyTree.addFirstChild(nodoM, 'P'); //Hijo de M


            /*Recorre el arbol en preOrden del Arbol (de la teoria)*/
                System.out.print("* Recorrido preOrden: ");
                System.out.print(recorrerArbol(MyTree));
                System.out.println("\n");


            } catch (InvalidOperationException e1) {
                  e1.getMessage();
            } catch (EmptyTreeException e2) {
                  e2.getMessage();
            } catch (InvalidPositionException e3) {
                  e3.getMessage();
            }


/*Método Recorrer en PreOrden (Porque así esta implementado en el árbol)*/
      public static String recorrerArbol(Tree<Character> T) {
        Iterator<Character> iterator = T.iterator(); //Guarda un iterador de todos los
elementos del arbol en preorden


        System.out.print("[ ");
        while(iterator.hasNext()) {
            System.out.print(iterator.next() + " "); //next devuelve el primero y avanza
al que sigue
        }
        System.out.print("]");


        return " ";
    }
```

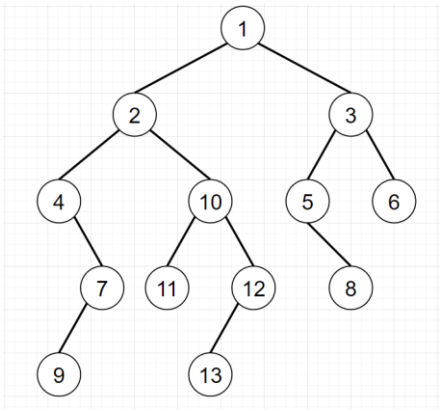
ÁRBOL BINARIO 1:




```
/*Árbol Binario de enteros*/
ArbolBinario<Integer> arbolB = new ArbolBinario<Integer>();

        try {

            Position<Integer> raiz = arbolB.createRoot(1); //Raiz.

            Position<Integer> nodo2 = arbolB.addLeft(raiz, 2); //Hijos de la raiz.
            Position<Integer> nodo4 = arbolB.addRight(raiz, 4);

            Position<Integer> nodo5 = arbolB.addLeft(nodo4, 5); //Hijos de 4.
            Position<Integer> nodo9 = arbolB.addRight(nodo4, 9);

            Position<Integer> nodo7 = arbolB.addRight(nodo5, 7); //Hijo de 5.

            Position<Integer> nodo10 = arbolB.addLeft(nodo9, 10); //Hijos de 9.
            Position<Integer> nodo44 = arbolB.addRight(nodo9, 4);

            Position<Integer> nodo8 = arbolB.addLeft(nodo2, 8); //Hijos de 2.
            Position<Integer> nodo6 = arbolB.addRight(nodo2, 6);

            Position<Integer> nodo3 = arbolB.addLeft(nodo6, 3); //Hijos de 6.
            Position<Integer> nodo11 = arbolB.addRight(nodo6, 1);
```

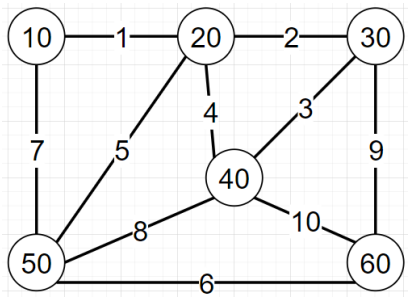
```
    System.out.println();

} catch (InvalidOperationException | InvalidPositionException elem) {
    elem.printStackTrace();
}
```

ARBOL BINARIO 2:




```
/*Arbol Binario de enteros*/
BinaryTree<Integer> arbol = new ArbolBinario<Integer>();

        try {

            Position<Integer> raiz = arbol.createRoot(1);
            Position<Integer> nodo2 = arbol.addLeft(raiz, 2);
            BTNode<Integer> nodo3 = (BTNode<Integer>) arbol.addRight(raiz, 3);
            BTNode<Integer> nodo4 = (BTNode<Integer>) arbol.addLeft(nodo2, 4);
            BTNode<Integer> nodo5 = (BTNode<Integer>) arbol.addLeft(nodo3, 5);
            BTNode<Integer> nodo6 = (BTNode<Integer>) arbol.addRight(nodo3, 6);
            BTNode<Integer> nodo7 = (BTNode<Integer>) arbol.addRight(nodo4, 7);
            BTNode<Integer> nodo8 = (BTNode<Integer>) arbol.addRight(nodo5, 8);
            BTNode<Integer> nodo9 = (BTNode<Integer>) arbol.addLeft(nodo7, 9);
            BTNode<Integer> nodo10 = (BTNode<Integer>) arbol.addRight(nodo2, 10);
            BTNode<Integer> nodo11 = (BTNode<Integer>) arbol.addLeft(nodo10, 11);
            BTNode<Integer> nodo12 = (BTNode<Integer>) arbol.addRight(nodo10, 12);
            BTNode<Integer> nodo13 = (BTNode<Integer>) arbol.addLeft(nodo12, 13);

            System.out.print("\n");

        } catch (InvalidOperationException | InvalidPositionException elem) {
            elem.printStackTrace();
        }
```

GRAFO:




```
/*Vértices de enteros, arcos de enteros*/

GrafoListaAdyacencia<Integer, Integer> grafo = new GrafoListaAdyacencia<Integer,
Integer>();


            try {


                  /*Cargar el grafo*/

                  Vertex<Integer> va = grafo.insertVertex(10);

                  Vertex<Integer> vb = grafo.insertVertex(20);

                  Vertex<Integer> vc = grafo.insertVertex(30);

                  Vertex<Integer> vd = grafo.insertVertex(40);

                  Vertex<Integer> ve = grafo.insertVertex(50);

                  Vertex<Integer> vf = grafo.insertVertex(60);

                  grafo.insertEdge(va, vb, 1);

                  grafo.insertEdge(vb, vc, 2);

                  grafo.insertEdge(vc, vd, 3);

                  grafo.insertEdge(vb, vd, 4);

                  grafo.insertEdge(vb, ve, 5);

                  grafo.insertEdge(ve, vf, 6);

                  grafo.insertEdge(va, ve, 7);

                  grafo.insertEdge(ve, vd, 8);

                  grafo.insertEdge(vf, vc, 9);

                  grafo.insertEdge(vf, vd, 10);


                  System.out.println();


                  /*Imprimir el grafo*/
```

```
System.out.println("Grafo: " );
grafo.DFS();

} catch (InvalidVertexException e1) {
      e1.printStackTrace();
}
```