```java
/**
 * MyBST.java
 * A basic structure for a Binary Search Tree (BST).
 *
 * Students will implement the insert, search, and traversal methods
 * for storing and retrieving integer data in a sorted, efficient structure.
 *
 * Key operations:
 * - insert(int value): adds a new value in order
 * - contains(int value): checks if a value exists
 * - inOrderPrint(): prints all values in sorted order
 *
 * This class does not support deletion.
 */
public class MyBST <T extends Comparable>{

    /**
     * Node represents a single element in the BST.
     * Each node may have a left and/or right child.
     */
    private static class Node<T> {
        T data;
        Node<T> left, right;

        public Node(T data) {
            this.data = data;
        }
    }

    // The root node of the tree
    private Node<T> root;

    /**
     * Public method to insert a value into the BST.
     * Uses a recursive helper method to place the value
     * in the correct position based on BST rules.
     *
     * @param value the integer to be inserted
     */
    public void insert(T value) {
        // TODO: Call insertHelper with root and value
        root = insertHelper(root, value);
    }

    /**
     * Recursive helper method for inserting a value into the tree.
     * If the current node is null, this is the correct insertion point.
     * If the value is less than the current node's data, recurse left.
     * If the value is greater, recurse right.
     *
     * @param node the current node in the traversal
     * @param value the value to insert
     * @return the updated node after insertion
     */
    private Node<T> insertHelper(Node<T> node, T value) {
        // TODO: Implement insert logic
        if (node == null){
            return new Node(value);
        }
        if (value.compareTo(node.data) < 0){
            node.left = insertHelper(node.left, value);
        }
        else {
            node.right = insertHelper(node.right, value);
        }
        return node;
    }

```

```
69      /**
70       * Determines whether the BST contains the specified value.
71       * Traverses the tree using a while loop, comparing at each step.
72       *
73       * @param value the value to search for
74       * @return true if the value exists in the tree, false otherwise
75       */
76      public boolean contains(T value) {
77          // TODO: Implement a while-loop search starting from root
78          Node<T> current_node = root;
79
80          while (current_node != null){
81              if (value.compareTo(current_node.data) == 0){
82                 return true;
83              }
84              else if (value.compareTo(current_node.data) < 0){
85                 current_node = current_node.left;
86              }
87
88              else {
89                 current_node = current_node.right;
90              }
91          }
92
93          return false;
94      }
95
96      /**
97       * Initiates an in-order traversal of the BST.
98       * Values will be printed from smallest to largest.
99       */
100     public void inOrderPrint() {
101         // TODO: Call inOrderHelper with root
102         inOrderHelper(root);
103     }
104
105     /**
106      * Performs in-order traversal of the BST.
107      * Visit order: left subtree, current node, right subtree.
108      *
109      * @param node the current node being visited
110      */
111     private void inOrderHelper(Node<T> node) {
112         // TODO: Implement in-order traversal
113         if (node == null){
114             return;
115         }
116
117         inOrderHelper(node.left);
118         System.out.print(node.data + " ");
119         inOrderHelper(node.right);
120     }
121
122     private void printTree(){
123          printTreeHelper(root, 0);
124     }
125
126     private void printTreeHelper(Node<T> node, int depth){
127          if (node == null){
128             return;
129          }
130
131          printTreeHelper(node.right, depth + 1);
132
133          for (int i = 0; i < depth; i++){
134             System.out.print("    ");
135          }
136
```

```
137          System.out.println(node.data);
138          printTreeHelper(node.left, depth + 1);
139     }
140
141     public static void main(String[] args){
142          MyBST<Integer> mBST = new MyBST<Integer>();
143          mBST.insert(5);
144          mBST.insert(7);
145          mBST.insert(3);
146          mBST.insert(2);
147          mBST.insert(1);
148          mBST.insert(8);
149          mBST.insert(6);
150          mBST.insert(4);
151          mBST.insert(10);
152
153          //mBST.inOrderPrint();
154          mBST.printTree();
155          System.out.println(mBST.contains(1));
156     }
157 }
```