Project Description: Binary Search Tree Implementation

In this project, you will implement a generic Binary Search Tree (BST) in Java. A BST is a tree data structure in which each node has at most two children and where the left subtree of a node contains only values less than the node's value, while the right subtree contains only values greater than the node's value. You are provided with the starter code with method stubs. Your task is to complete each method based on the specifications below.

Method Requirements

1. **`isEmpty()`**
   Purpose: Determine if the BST is empty.
   Description:
   - Check if the tree's root is `null`.
   - Return `true` if there are no nodes (i.e., `root` is `null`), otherwise return `false`.
2. **`max()`**
   Purpose: Retrieve the maximum element in the BST.
   Description:
   - Starting from the root, continuously traverse to the right child because in a BST the right child always holds a greater value than its parent.
   - Once you reach a node with no right child, return the data from that node as it is the maximum element in the tree.
3. **`contains(E obj)`**
   Purpose: Check if the BST contains a given object.
   Description:
   - Start from the root and compare the target object with the current node's data.
   - If they match, return `true`.
   - If the target is less than the current node's data, search in the left subtree.
   - If the target is greater, search in the right subtree.
   - If you reach a `null` node, the object is not in the tree; return `false`.
4. **`add(E obj)`**
   Purpose: Insert a new element into the BST.
   Description:
   - If the BST is empty, create a new node and assign it to `root`.

- Otherwise, traverse the tree comparing the new element with the current node's data.
- Move left if the new element is less than the current node; move right if it is greater.
- When an appropriate `null` position is found, insert the new node so that the BST properties remain intact.

5. **`size()`**
   Purpose: Count the total number of nodes in the BST.
   Description:
   - Implement this method using recursion.
   - For each node, count it (1) plus the number of nodes in its left subtree and the number of nodes in its right subtree.
   - If a node is `null`, return 0.

6. **`inOrder()`**
   Purpose: Traverse the BST in in-order (left subtree, node, right subtree) and print the nodes.
   Description:
   - In-order traversal of a BST prints the elements in sorted order.
   - Use recursion: first traverse the left subtree, then process (print) the current node's data, and finally traverse the right subtree.

7. **`preOrder()`**
   Purpose: Traverse the BST in pre-order (node, left subtree, right subtree) and print the nodes.
   Description:
   - In pre-order traversal, print the current node's data first, then traverse the left subtree, and finally traverse the right subtree.
   - Use recursion to implement this method.