|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input** | **Expected Behavior** | **Aspect Tested** |
| **Balanced BST** | (53 (28 (11 \* \*) (41 \* \*)) (83 (67 \* \*) \*)) | Prints indented tree; message: “It is a balanced binary search tree” | Indentation, BST detection, balance detection |
| **Unbalanced BST** | (63 (51 (20 (13 \* \*) \*) \*) \*) | Prints original; “It is a binary search tree but it is not balanced”; prints rebuilt tree; shows original height 3, new 2 | Rebuilding, height calc, unbalanced detection |
| **Not a BST** | (13 (53 \* \*) (11 (59 \* \*) \*)) | Prints original; “It is not a binary search tree”; prints rebuilt tree; heights (both 2) | BST violation handling, rebuilding |
| **Incomplete Tree** | (53 (28 \* \*) | Error: “Missing right parenthesis” | Syntax: missing ) |
| **Data Not Integer** | (a ( \* \*) ( \* \*)) | Error: “Data is not an integer” | Syntax: non‐integer data |
| **Extra Characters at End** | (53 \* \*) extra | Error: “Extra characters at the end” | Syntax: trailing input |
| **Missing Left Parenthesis** | 53 (28 \* \*) ( \* \*) | Error: “Missing left parenthesis” | Syntax: missing leading ( |
| **Single Node Tree** | (42 \* \*) | Prints 42; “It is a balanced binary search tree” | Leaf case, minimal tree |

A screenshot of a computer program

AI-generated content may be incorrect.

┌─────────────────────────────────────────────────────────────┐  
│ BinaryTree │  
├─────────────────────────────────────────────────────────────┤  
│ - Node root │  
├─────────────────────────────────────────────────────────────┤  
│ + BinaryTree(String) throws InvalidSyntaxException │  
│ + BinaryTree(List<Integer>) │  
│ + String toIndentedString() │  
│ + boolean isBST() │  
│ + boolean isBalanced() │  
│ + int getHeight() │  
│ + List<Integer> getValues() │  
├─────────────────────────────────────────────────────────────┤  
│ ── private static class Node │  
│ - int value │  
│ - Node left, right │  
└─────────────────────────────────────────────────────────────┘  
  
┌──────────────────────────────────┐  
│ InvalidSyntaxException │  
├──────────────────────────────────┤  
│ extends Exception │  
├──────────────────────────────────┤  
│ + InvalidSyntaxException(String) │  
└──────────────────────────────────┘  
  
┌────────────────────────────────┐  
│ Project3 │  
├────────────────────────────────┤  
│ + static void main(String[]) │  
└────────────────────────────────┘

**Lessons Learned**

Designing this project reinforced how critical a solid, immutable data‐structure foundation is for building reliable software. By parsing a nested string representation, I honed my attention to detail around syntax and error handling—important for any production‐grade parser. Constructing balanced trees from sorted data illustrated how powerful divide‐and‐conquer patterns can be. As a professional engineer with an entrepreneurial mindset, I appreciated the value of clean separation of concerns: parsing, core logic, I/O, and exception types each live in their own class. Finally, writing a thorough test plan and clear UML diagram reminded me that good documentation accelerates collaboration and maintenance—key lessons for any business or startup I might found in the future.