Project 1
Binary Search Tree

Made by student
2IiE-SP
Volodymyr Chuchka

BinarySearchTree Class:

Fields:

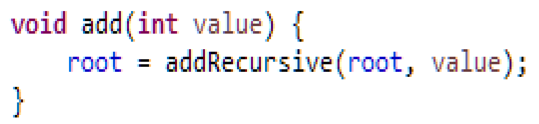
>>>>>>`Node root`: Represents the root node of the binary search tree<<<<<<

Constructor:

>>>>>>`BinarySearchTree()`: Initializes an empty binary search tree.<<<<<<

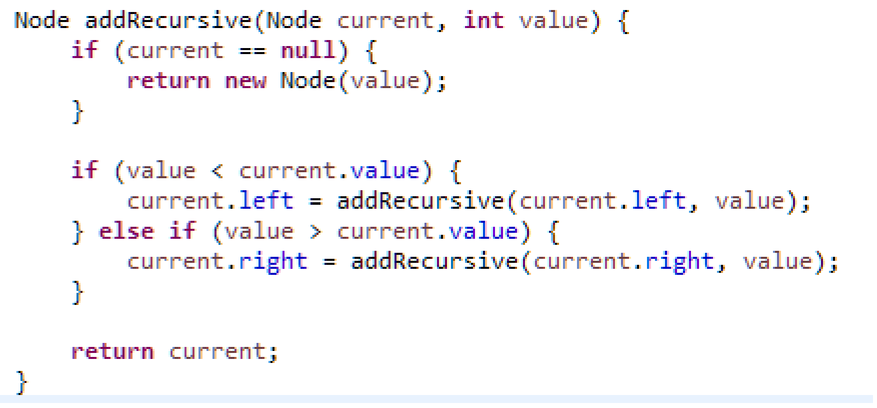
Methods:

1. `void add(int value)`:
  - Description: Adds a new node with the given value to the binary search tree.
  - Parameters: `int value` - the value to be added.



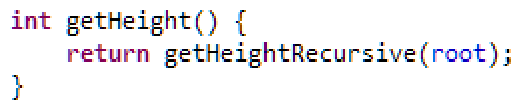
```
void add(int value) {
    root = addRecursive(root, value);
}
```

2. `Node addRecursive(Node current, int value)`:
  - Description: Helper method for recursive insertion of a node with the given value into the binary search tree.
  - Parameters:
    - `Node current` - the current node in the recursion.
    - `int value` - the value to be added.
  - Returns: The updated tree with the new node.



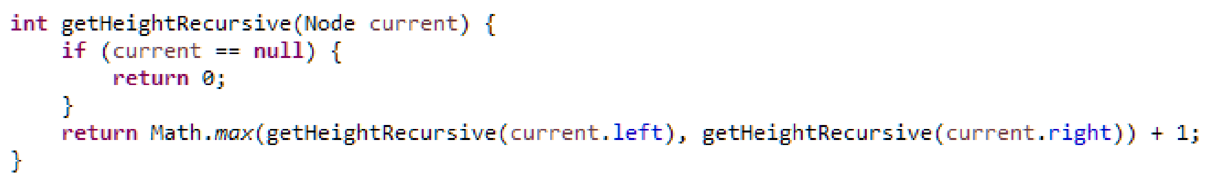
```
Node addRecursive(Node current, int value) {
    if (current == null) {
        return new Node(value);
    }

    if (value < current.value) {
        current.left = addRecursive(current.left, value);
    } else if (value > current.value) {
        current.right = addRecursive(current.right, value);
    }

    return current;
}
```

3. `int getHeight()`:
  - Description: Calculates the height of the binary search tree.
  - Returns: The height of the tree.



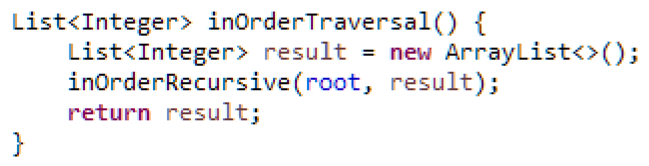
```
int getHeight() {
    return getHeightRecursive(root);
}
```

4. `int getHeightRecursive(Node current)`:
   - Description: Helper method for recursive calculation of the height of the binary search tree.
   - Parameters: `Node current` - the current node in the recursion.
   - Returns: The height of the subtree rooted at the current node.



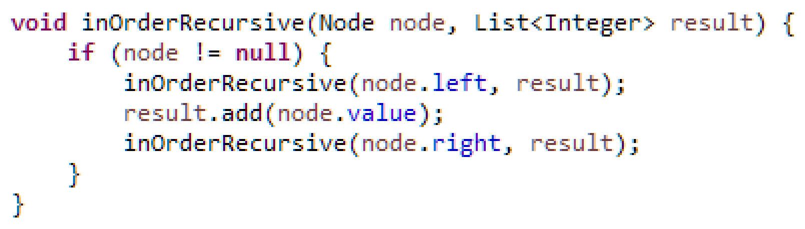
```
int getHeightRecursive(Node current) {
    if (current == null) {
        return 0;
    }
    return Math.max(getHeightRecursive(current.left), getHeightRecursive(current.right)) + 1;
}
```

5. `List<Integer> inOrderTraversal()`:
   - Description: Performs an in-order traversal of the binary search tree.
   - Returns: A list of integers representing the elements in the tree in sorted order.



```
List<Integer> inOrderTraversal() {
    List<Integer> result = new ArrayList<>();
    inOrderRecursive(root, result);
    return result;
}
```

6. `void inOrderRecursive(Node node, List<Integer> result)`:
   - Description: Helper method for recursive in-order traversal.
   - Parameters:
     - `Node node` - the current node in the recursion.
     - `List<Integer> result` - the list to store the in-order traversal result.

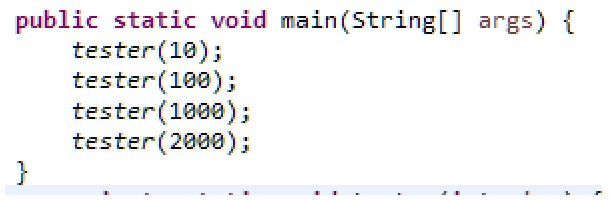


```
void inOrderRecursive(Node node, List<Integer> result) {
    if (node != null) {
        inOrderRecursive(node.left, result);
        result.add(node.value);
        inOrderRecursive(node.right, result);
    }
}
```

Main Class:

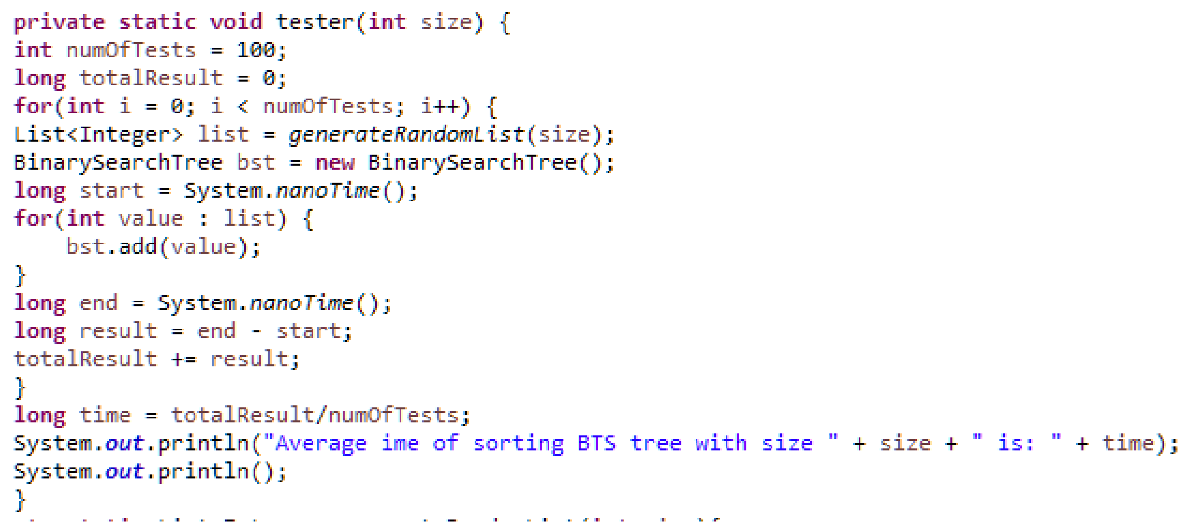
Methods:

1. `public static void main(String[] args)`:
   - Description: The main entry point of the program. Calls the `tester` method with different tree sizes.



```
public static void main(String[] args) {
    tester(10);
    tester(100);
    tester(1000);
    tester(2000);
}
```

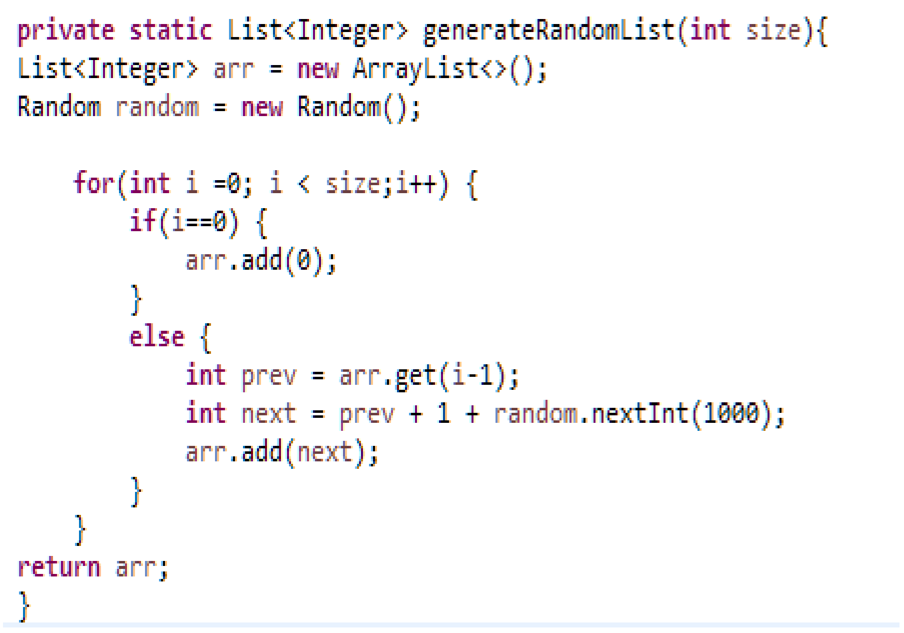
2. `private static void tester(int size)`:
   - Description: Tests the performance of the binary search tree by measuring the time it takes to add elements from a randomly generated list.
   - Parameters: `int size` - the size of the randomly generated list and amount of numbers in our BST tree.



```
private static void tester(int size) {
int numOfTests = 100;
long totalResult = 0;
for(int i = 0; i < numOfTests; i++) {
List<Integer> list = generateRandomList(size);
BinarySearchTree bst = new BinarySearchTree();
long start = System.nanoTime();
for(int value : list) {
    bst.add(value);
}
long end = System.nanoTime();
long result = end - start;
totalResult += result;
}
long time = totalResult/numOfTests;
System.out.println("Average ime of sorting BTS tree with size " + size + " is: " + time);
System.out.println();
}
```

PS: It takes list with random unrepeatable numbers and fill with them our BST tree and give to us average time of sorting 100 different tabbles.

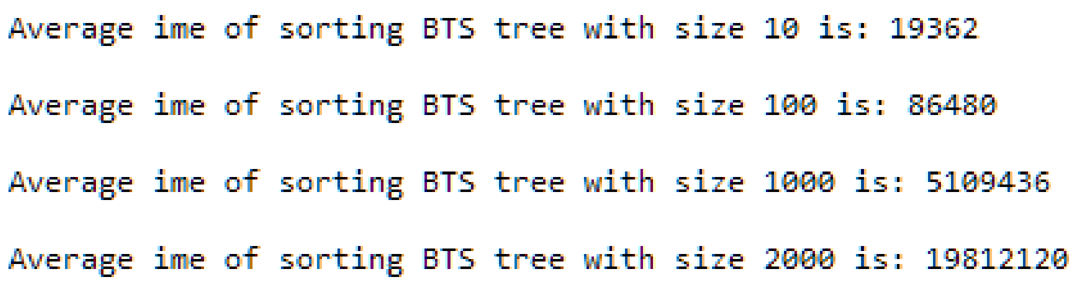
3. `private static List<Integer> generateRandomList(int size)`:
   - Description: Generates a random list of integers with a specified size.
   - Parameters: `int size` - the size of the list.
   - Returns: A randomly generated list of integers.



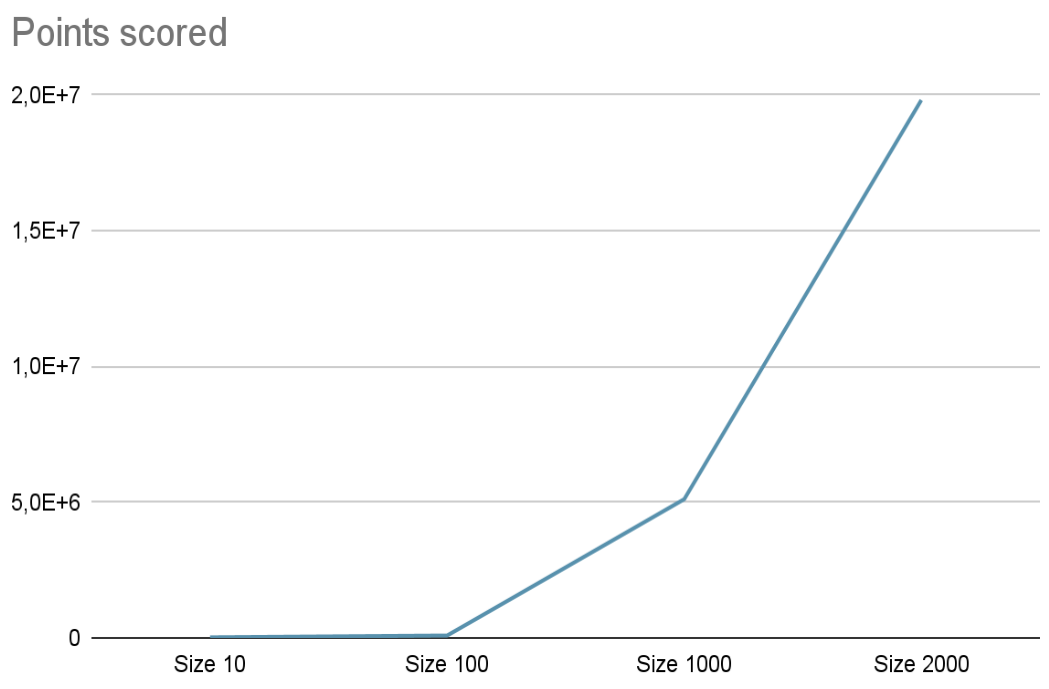
```
private static List<Integer> generateRandomList(int size){
List<Integer> arr = new ArrayList<>();
Random random = new Random();

    for(int i =0; i < size;i++) {
        if(i==0) {
            arr.add(0);
        }
        else {
            int prev = arr.get(i-1);
            int next = prev + 1 + random.nextInt(1000);
            arr.add(next);
        }
    }
return arr;
}
```

PS: prev + 1 + random.nextInt(1000) is made for filling table with unrepeatable numbers

RESULT OF ANALYZING TIME OF SORTING



```
Average ime of sorting BTS tree with size 10 is: 19362

Average ime of sorting BTS tree with size 100 is: 86480

Average ime of sorting BTS tree with size 1000 is: 5109436

Average ime of sorting BTS tree with size 2000 is: 19812120
```


Points scored
2,0E+7
1,5E+7
1,0E+7
5,0E+6
Size 10
Size 100
Size 1000
Size 2000