Untitled

HTML Content

1. **Java autoboxing and equals()**. Consider two double values a and b and their corresponding Double values x and y. 1 1

- Find values such that (a == b) is true but x.equals(y) is false.
- Find values such that (a == b) is false but x.equals(y) is true.

Note: these interview questions are ungraded and purely for your own enrichment. To get a hint, submit a solution.

```
1. a=0.0, b=-0.0
2. a=b=Double.NaN
```

Correct

Hint: IEEE floating point arithmetic has some peculiar rules for 0.0, −0.0, and NaN. Java requires that equals() implements an equivalence relation.

2. **Check if a binary tree is a BST.** Given a binary tree where each Node contains a key, determine whether it is a binary search tree. Use extra space proportional to the height of the tree. 1 1

```
private class Node {
private int key; // sorted by key
private int val; // associated data
private Node left, right; // left and right subtrees
private int N; // number of nodes in subtree

public Node(int key, int val, int N) {
this.key = key;
this.val = val;
this.N = N;
}
}

public boolean checkBST(Node p, int min, int max) {
if (p == null) return true;
if(p.key >= max || p.key <= min ) return false;
return checkBST(p.left, min, p.key) && checkBST(p.right, p.key, max);
}
```

Correct

Hint: design a recursive function $\text{isBST}(\text{Node x}, \text{Key min}, \text{Key max})$ that determines whether x is the root of a binary search tree with all keys between min and max.

3.Inorder traversal with constant extra space. Design an algorithm to perform an inorder traversal of a binary search tree using only a constant amount of extra space. 1 1

```
public void inorder(Node root) {
if (root == null) return;

Node previous;
Node current = root;
while (current != null) {
//current has no left child, print current, then go right
if (current.left == null) {
System.out.println(current.val);
current = current.right;
}
else {
previous = current.left;

//go down to current left children's rightmost child
while (previous.right != null && previous.right != current) {
previous = previous.right;
}

//if the rightmost child hasn't being linked to current, then link it, and traverse to current left
if (previous.right == null) {
previous.right = current;
current = current.left;
}
//if the rightmost child already linked to current (current left children being traversed), then print current and cut the link to restore tree structure
else {
previous.right = null;
System.out.println(current.val);
current = current.right;
}
}
}
}
```

Correct

Hint: you may modify the BST during the traversal provided you restore it upon completion.

4. **Web tracking.** Suppose that you are tracking n web sites and m users and you want to support the following API: 1 1

- User visits a website.
- How many times has a given user visited a given site?

What data structure or data structures would you use?

```
public static void main(String[] args) {
double a = Double.NaN;
double b = Double.NaN;
Double x = new Double(a);
Double y = new Double(b);
System.out.println(a==b);
System.out.println(x.equals(y));
}
```

Correct

Hint: maintain a symbol table of symbol tables.