**Q4>** We are having two implementations of the binary search tree. In the Composite implementation, we have a class for empty node i.e. EmptyNode and a different class for non-empty nodes i.e. ElementNode. The methods which are shared across these two implementations are defined in an interface called Node, which allows to have uniformity. This approach allows us to take the advantage of Java’s automatic dynamic dispatching to identify the type of node and call the respective method.

This automation is one of the advantages we can obtain using object oriented programming approach.

In the second approach, i.e. in single node approach we have just one type of node and in-order to differentiate between the empty and non-empty node, we write an extra piece of code which involves an if condition to check if node is null/empty and perform necessary action.

This approach is more like the one used in functional programming languages. This results in cluttering of code within the same function. The impact is more when we have many methods being defined.

The other difference between composite and single node implantation is that, we can traverse backwards using parent reference in case of composite, where-as in single node implementation the only option to traverse back is recursive stack frame.

In case of composite implementation, the numbers of methods increase with time whereas in case of single node implementation too many if cases may cause code cluttering.