## **UML** Binary tree

```
node
- key: int
- left: node*
- right: node*
+ node(key: int): constructor
                               BST
 - root: node*
                         <<constructor>>
 + BST()
                          <<destructor>>
 + ~BST()
                          <<interface>>
 - add_leaf_private(key: int, Ptr: node*): void
 - print in order private(Ptr: node*): void
 - return node private(key: int, Ptr: node*): node*
 - find smallest private(Ptr: node*): int
 - remove node private(key: int, parent: node*): void
 - remove root match(): void
 - remove match(parent: node*, match: node*, left: bool): void
 - create leaf(key: int): node*
 - return node(key: int): node*
 - remove_subtree(Ptr: node*): void
 + add leaf(key: int): void
 + print in order(): void
 + int return root key(): int
 + print children(key: int): void
 + int find smallest(): int
 + remove_node(key: int): void
```

## node - kevs: int\* - childPtrs: node\*\* - numKeys: int - t: int - leaf: bool + node (t pass: int, leaf: bool): constructor + print(): void + key\_search(key: int): node\* + deletion(key: int): void + int find key(key: int): int + insert not full(key: int): void + split the child(i: int, y: node\*): void + deletion\_from\_leaf(ind: int): void + deletion from not leaf(ind: int): void + get previous(ind: int): int + get next(ind: int): int + fill(ind: int): void + get\_from\_previous(ind: int): void + get from next(ind: int): void + merge(ind: int): void

BT

- root: node\*

+ print(): void

+ BT(t: int): **constructor** 

+ key\_search(key: int): node\*
+ add\_leaf(key: int): void
+ deletion(key: int): void