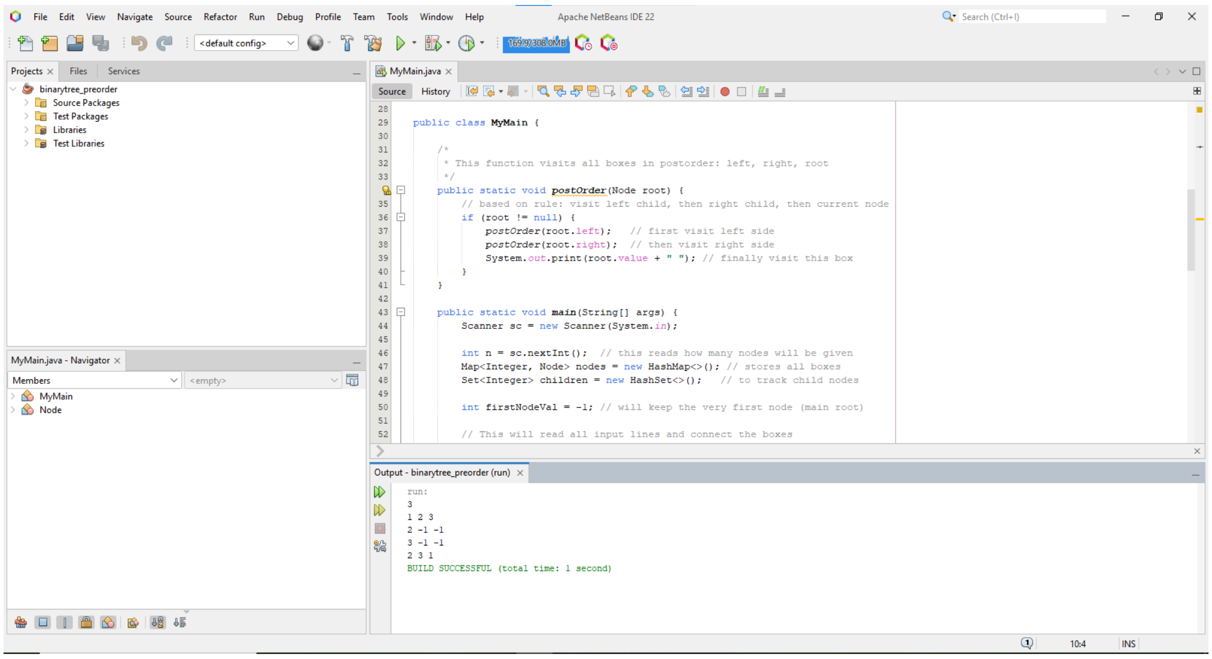
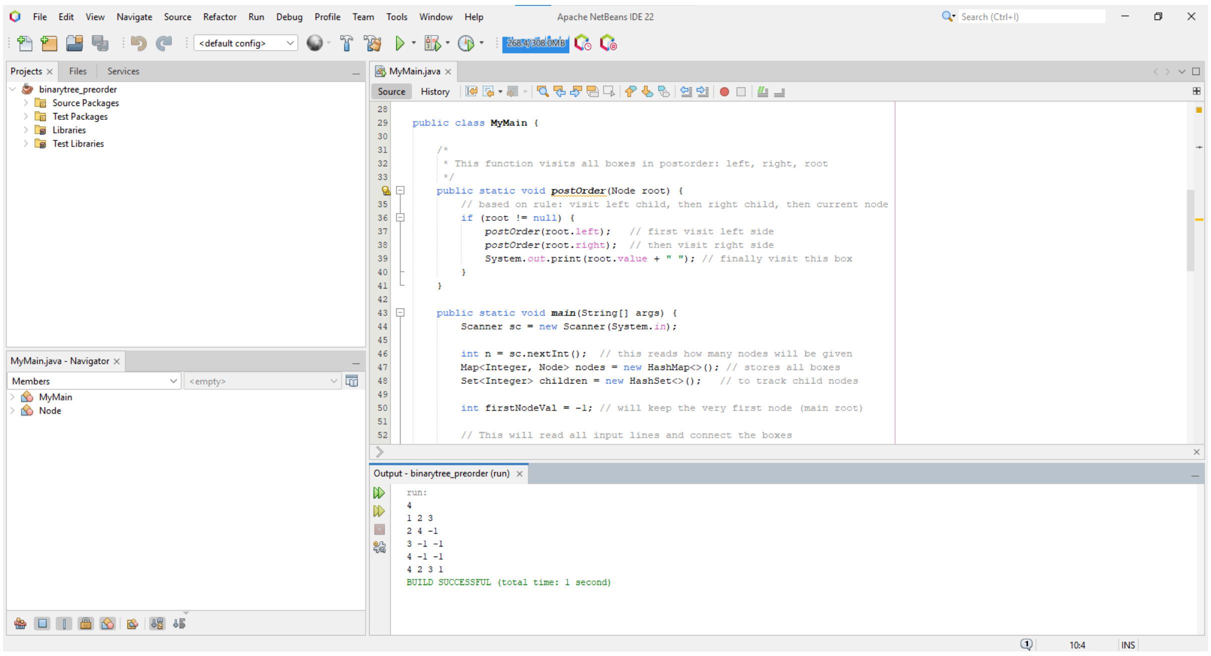


```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

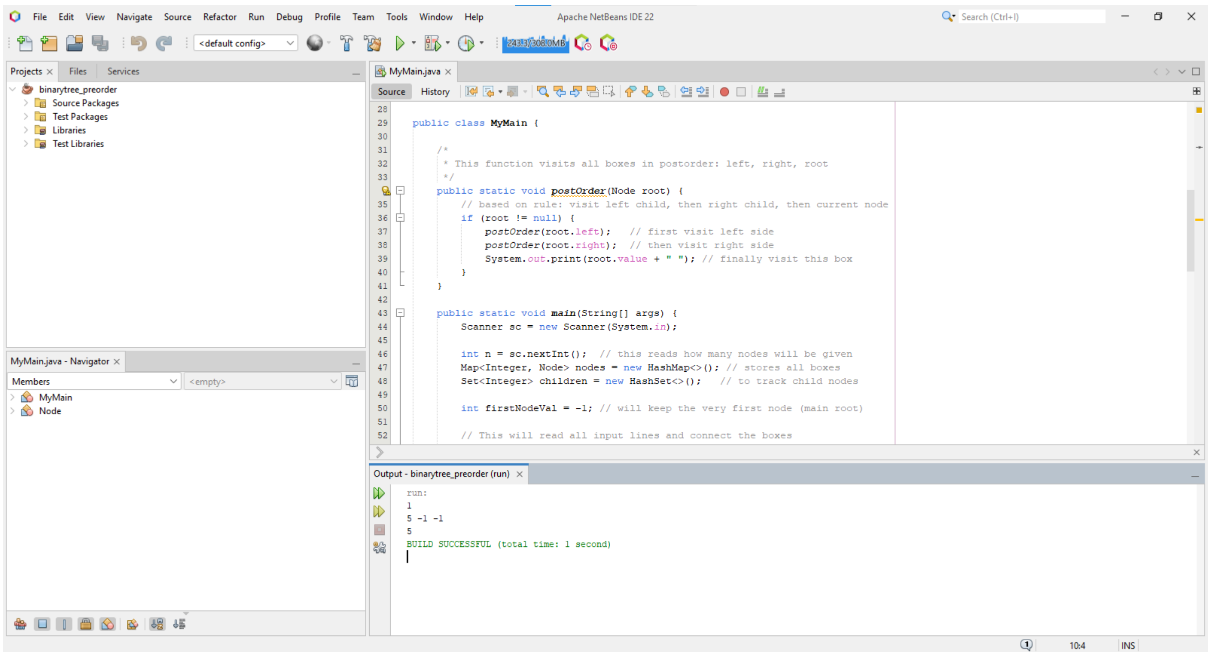
```
run:
3
1 2 3
2 -1 -1
3 -1 -1
2 3 1
BUILD SUCCESSFUL (total time: 1 second)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

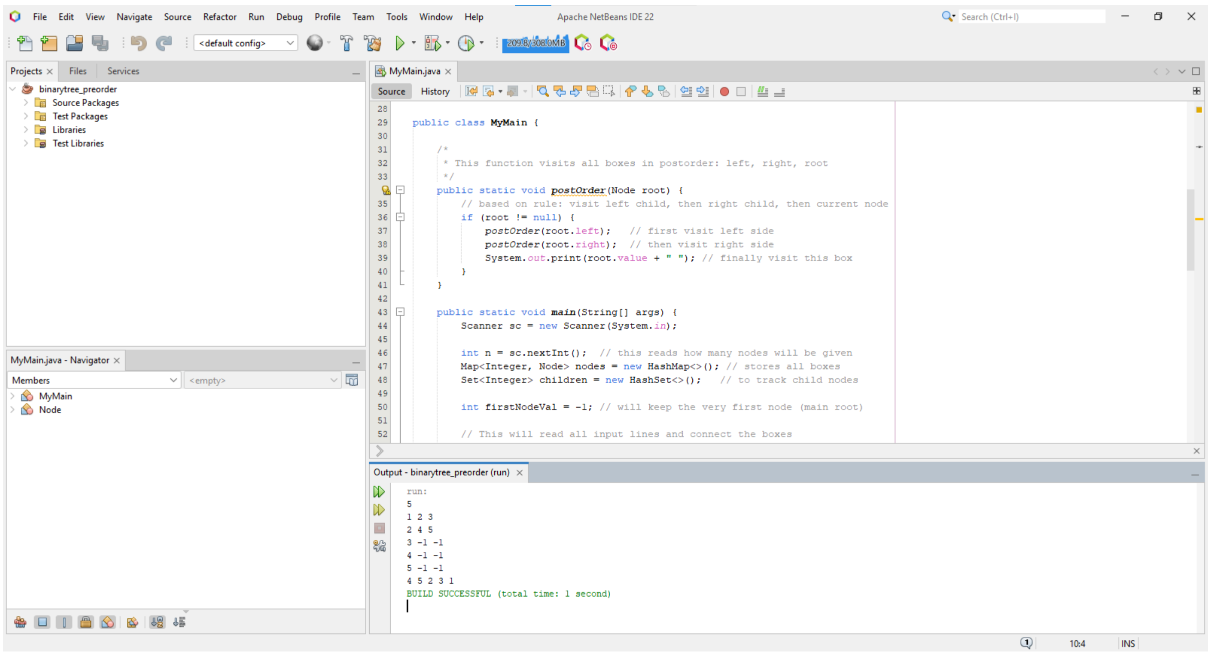
```
run:
4
1 2 3
2 4 -1
3 -1 -1
4 -1 -1
4 2 3 1
BUILD SUCCESSFUL (total time: 1 second)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

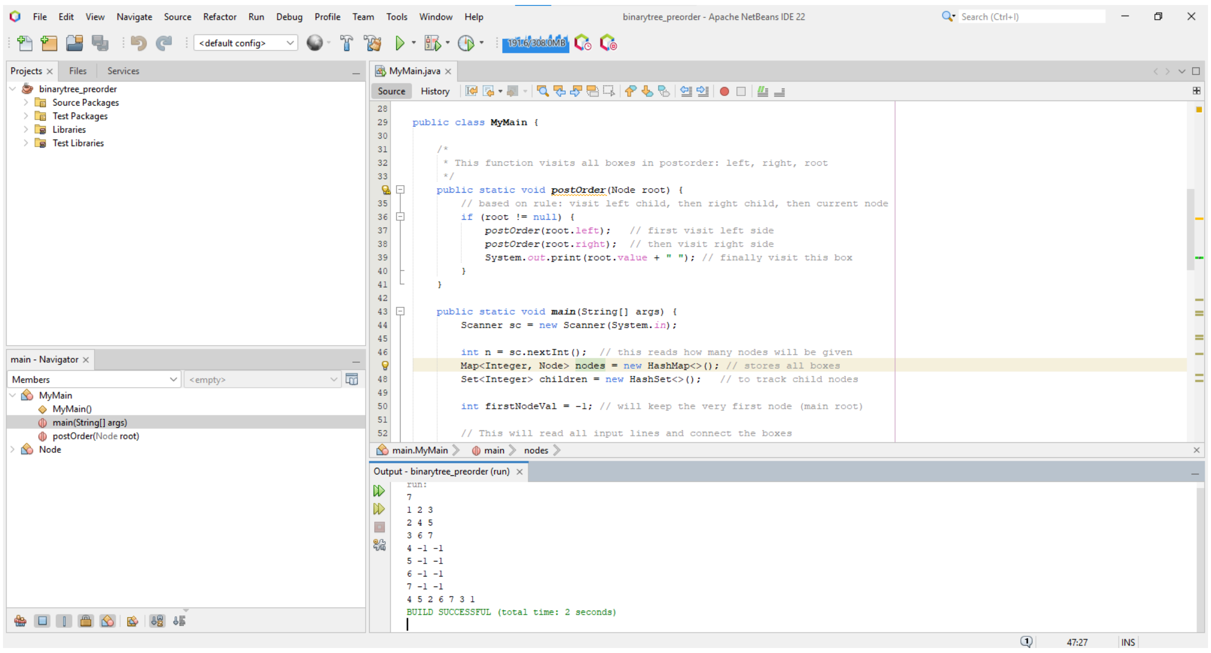
```
run:
1
5 -1 -1
5
BUILD SUCCESSFUL (total time: 1 second)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

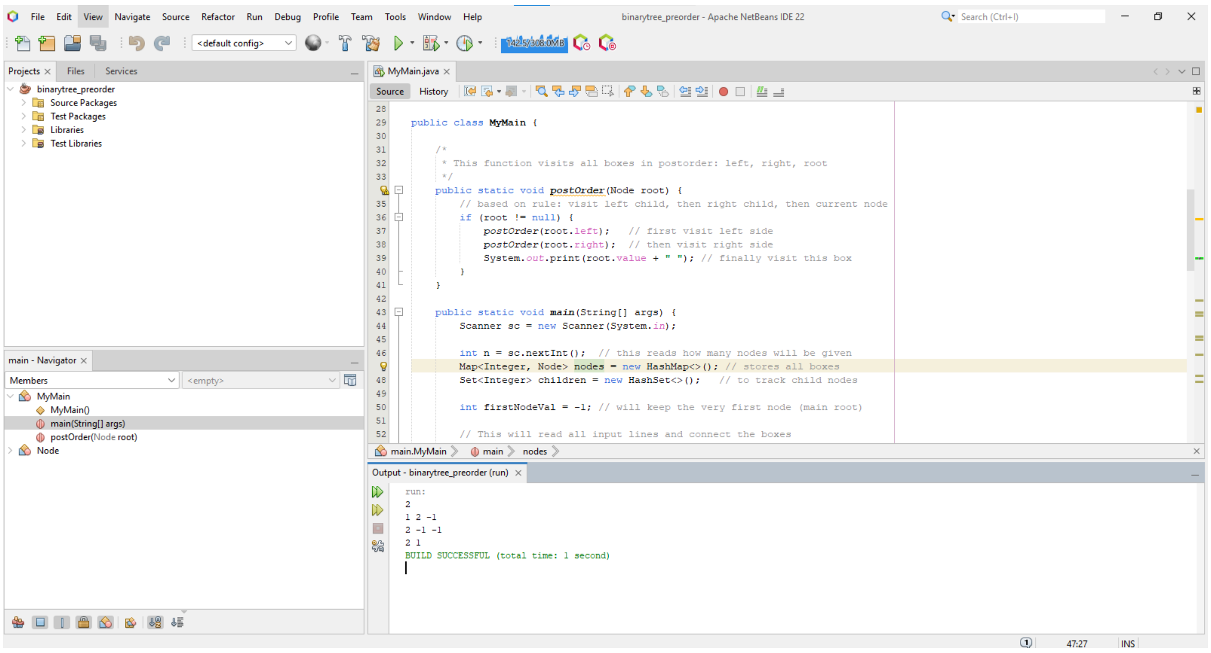
```
run:
5
1 2 3
2 4 5
3 -1 -1
4 -1 -1
5 -1 -1
4 5 2 3 1
BUILD SUCCESSFUL (total time: 1 second)
```



```java

public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

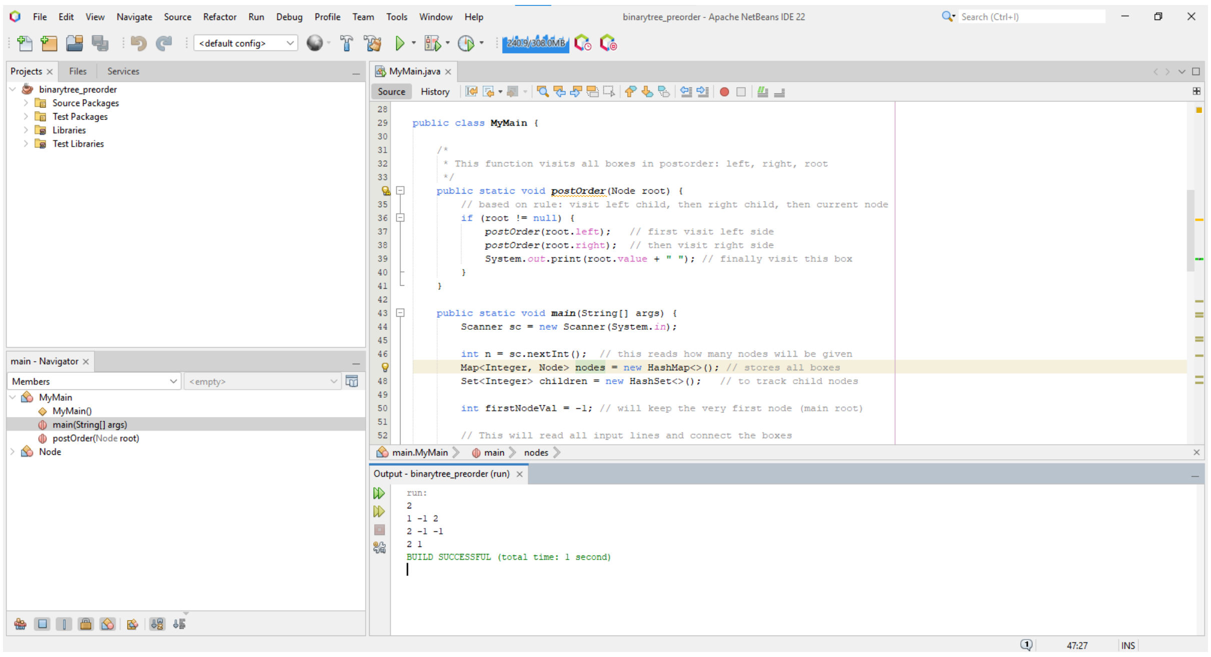
```
run:
7
1 2 3
2 4 5
3 6 7
4 -1 -1
5 -1 -1
6 -1 -1
7 -1 -1
4 5 2 6 7 3 1
BUILD SUCCESSFUL (total time: 2 seconds)
```



```java

public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

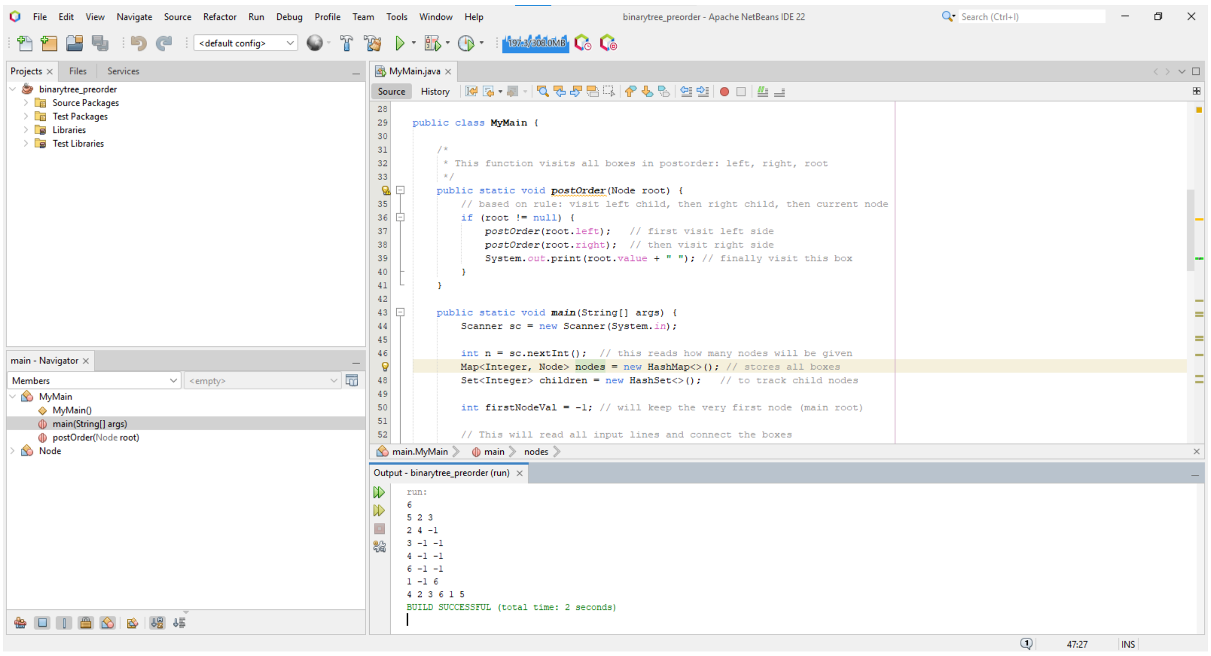
Output - binarytree_preorder (run)

```
run:
2
1 2 -1
2 -1 -1
2 1
BUILD SUCCESSFUL (total time: 1 second)
```



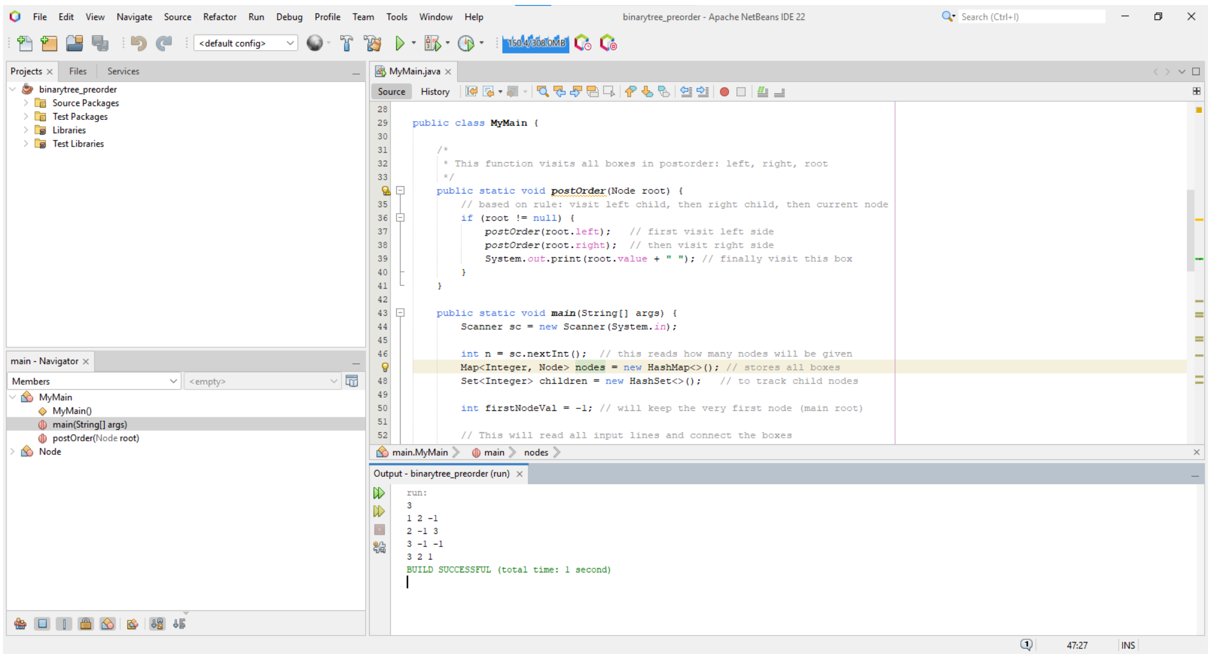
```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

```
run:
2
1 -1 2
2 -1 -1
2 1
BUILD SUCCESSFUL (total time: 1 second)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

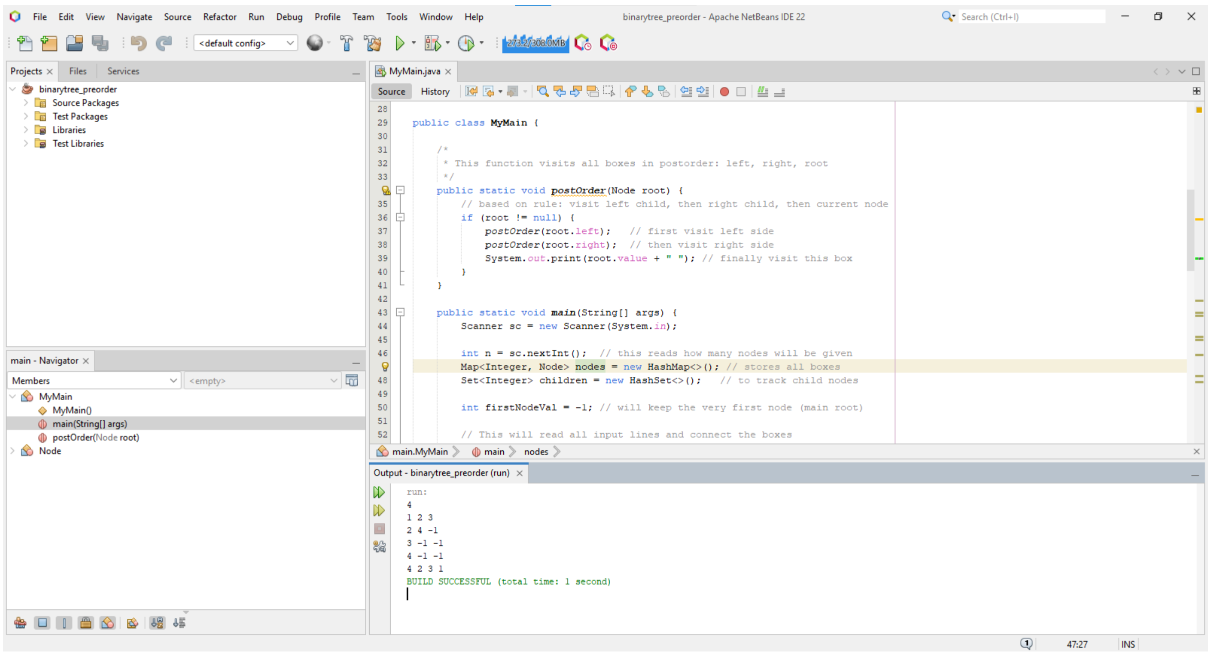
```
run:
6
5 2 3
2 4 -1
3 -1 -1
4 -1 -1
6 -1 -1
1 -1 6
4 2 3 6 1 5
BUILD SUCCESSFUL (total time: 2 seconds)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

```
run:
3
1 2 -1
2 -1 3
3 -1 -1
3 2 1
BUILD SUCCESSFUL (total time: 1 second)
```



```java
public class MyMain {

    /*
     * This function visits all boxes in postorder: left, right, root
     */
    public static void postOrder(Node root) {
        // based on rule: visit left child, then right child, then current node
        if (root != null) {
            postOrder(root.left);   // first visit left side
            postOrder(root.right);  // then visit right side
            System.out.print(root.value + " "); // finally visit this box
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();  // this reads how many nodes will be given
        Map<Integer, Node> nodes = new HashMap<>(); // stores all boxes
        Set<Integer> children = new HashSet<>();   // to track child nodes

        int firstNodeVal = -1; // will keep the very first node (main root)

        // This will read all input lines and connect the boxes
```

Output - binarytree_preorder (run)

```
run:
4
1 2 3
2 4 -1
3 -1 -1
4 -1 -1
4 2 3 1
BUILD SUCCESSFUL (total time: 1 second)
```