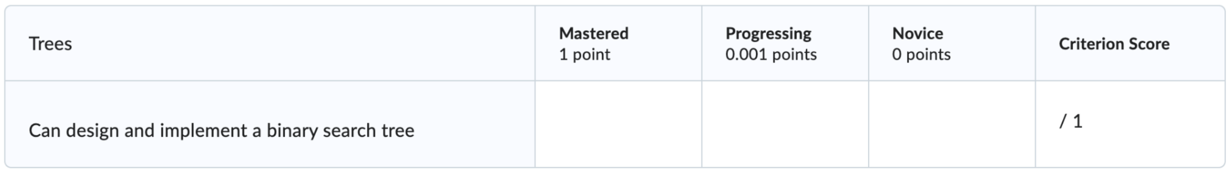
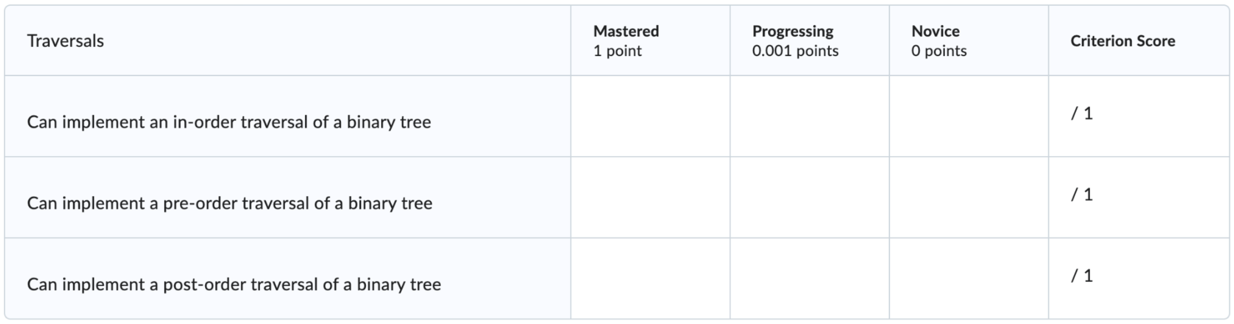
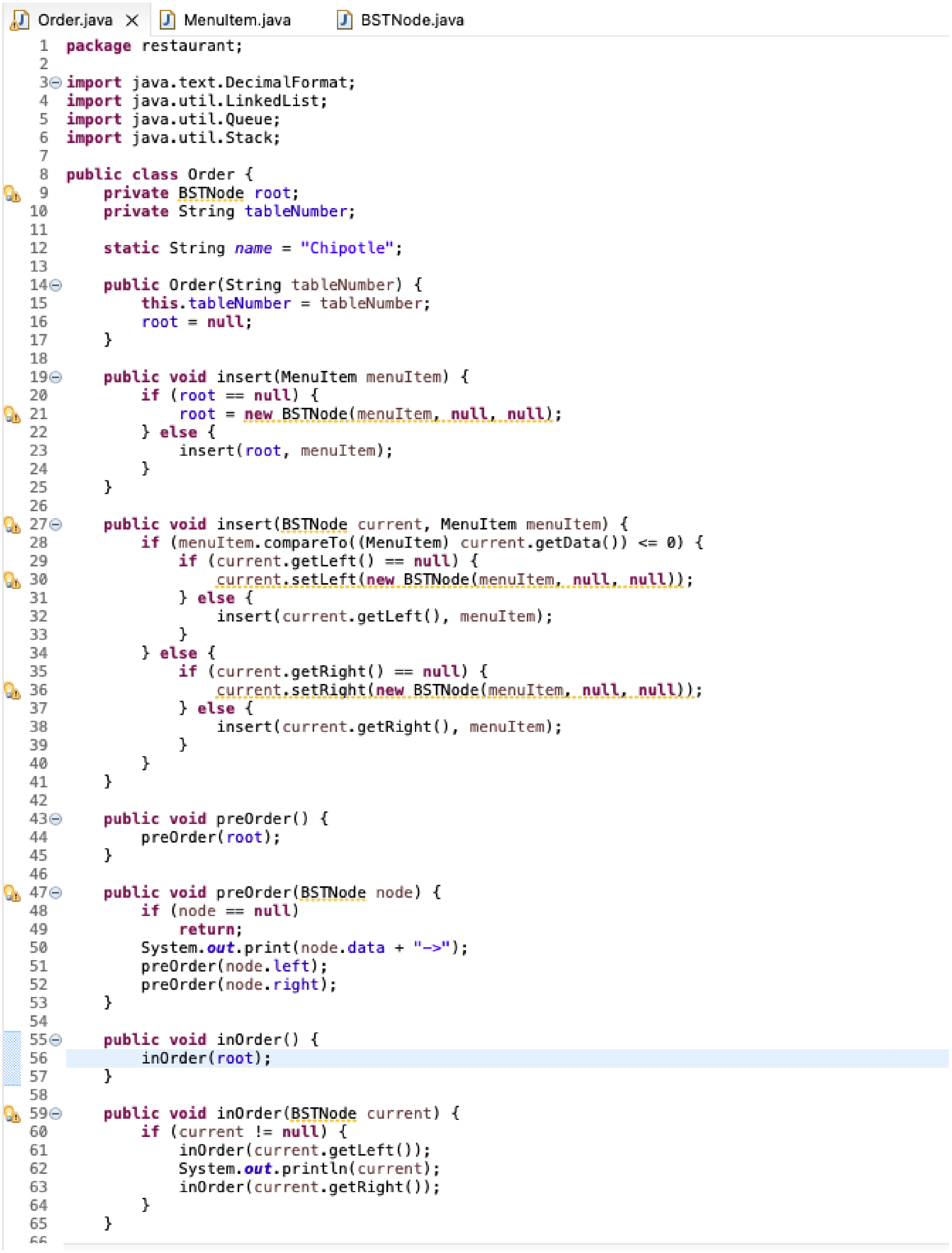
Traversals Of Binary Search Tree Competency #8



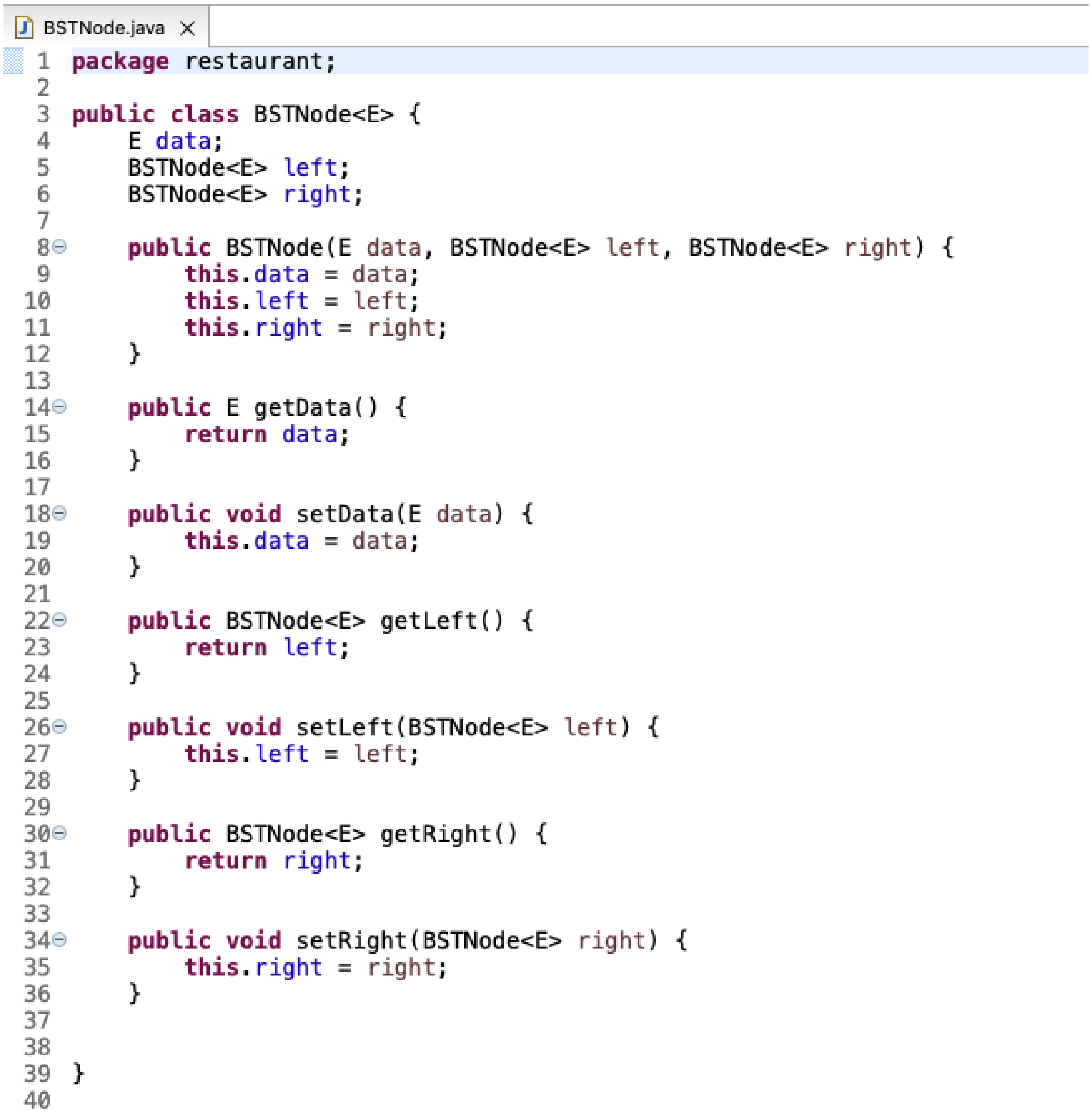
Trees	**Mastered** 1 point	**Progressing** 0.001 points	**Novice** 0 points	**Criterion Score**
Can design and implement a binary search tree				/ 1



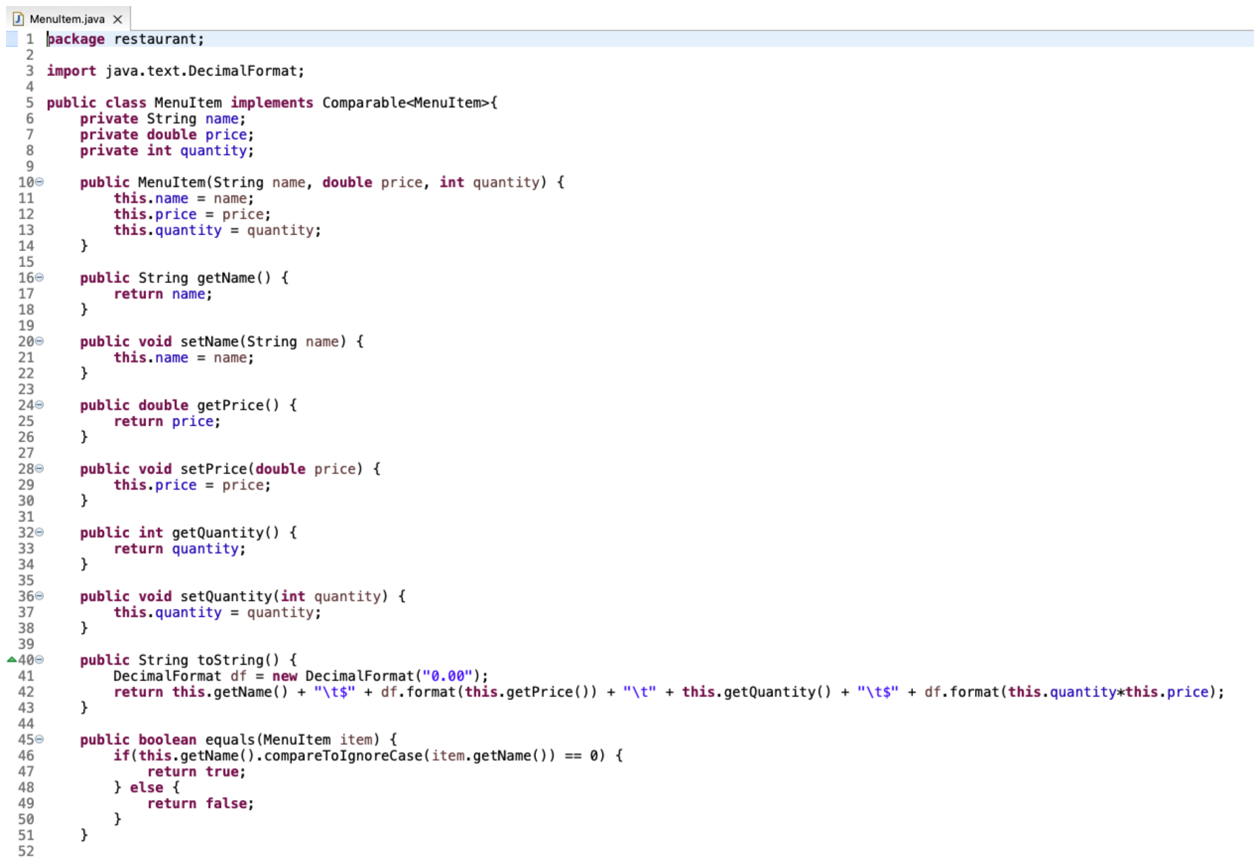
Traversals	**Mastered** 1 point	**Progressing** 0.001 points	**Novice** 0 points	**Criterion Score**
Can implement an in-order traversal of a binary tree				/ 1
Can implement a pre-order traversal of a binary tree				/ 1
Can implement a post-order traversal of a binary tree				/ 1



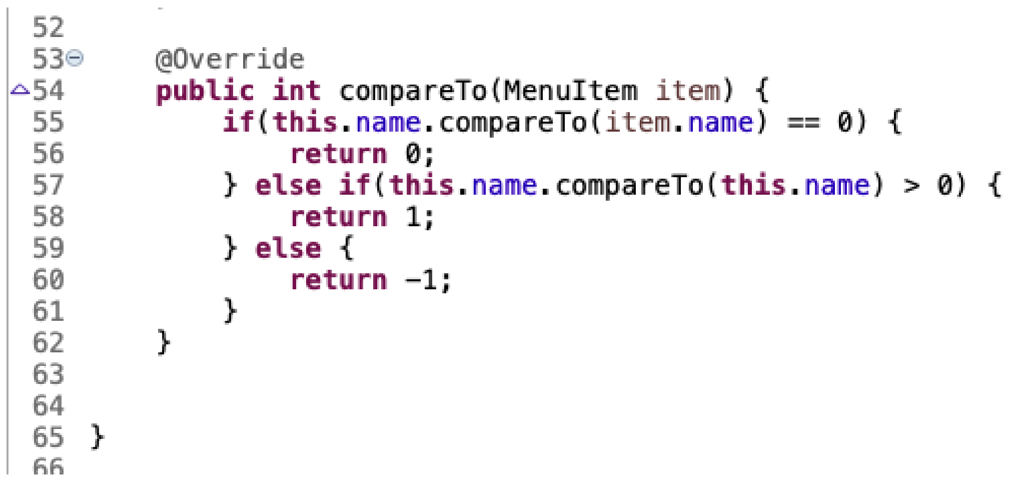
```java
package restaurant;

import java.text.DecimalFormat;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class Order {
    private BSTNode root;
    private String tableNumber;

    static String name = "Chipotle";

    public Order(String tableNumber) {
        this.tableNumber = tableNumber;
        root = null;
    }

    public void insert(MenuItem menuItem) {
        if (root == null) {
            root = new BSTNode(menuItem, null, null);
        } else {
            insert(root, menuItem);
        }
    }

    public void insert(BSTNode current, MenuItem menuItem) {
        if (menuItem.compareTo((MenuItem) current.getData()) <= 0) {
            if (current.getLeft() == null) {
                current.setLeft(new BSTNode(menuItem, null, null));
            } else {
                insert(current.getLeft(), menuItem);
            }
        } else {
            if (current.getRight() == null) {
                current.setRight(new BSTNode(menuItem, null, null));
            } else {
                insert(current.getRight(), menuItem);
            }
        }
    }

    public void preOrder() {
        preOrder(root);
    }

    public void preOrder(BSTNode node) {
        if (node == null)
            return;
        System.out.print(node.data + "->");
        preOrder(node.left);
        preOrder(node.right);
    }

    public void inOrder() {
        inOrder(root);
    }

    public void inOrder(BSTNode current) {
        if (current != null) {
            inOrder(current.getLeft());
            System.out.println(current);
            inOrder(current.getRight());
        }
    }
```


BSTNode.java

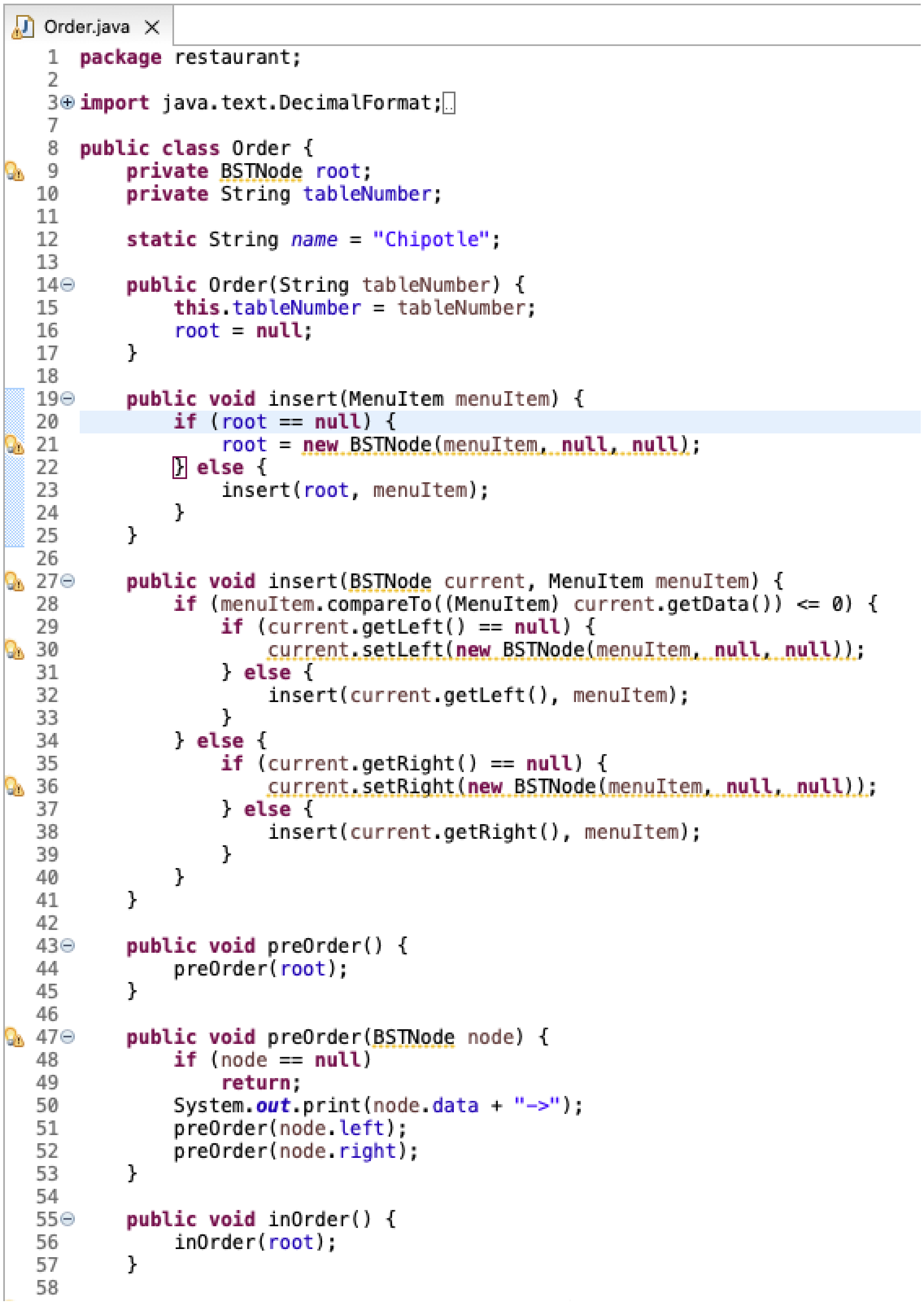
```java
package restaurant;

public class BSTNode<E> {
    E data;
    BSTNode<E> left;
    BSTNode<E> right;

    public BSTNode(E data, BSTNode<E> left, BSTNode<E> right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }

    public E getData() {
        return data;
    }

    public void setData(E data) {
        this.data = data;
    }

    public BSTNode<E> getLeft() {
        return left;
    }

    public void setLeft(BSTNode<E> left) {
        this.left = left;
    }

    public BSTNode<E> getRight() {
        return right;
    }

    public void setRight(BSTNode<E> right) {
        this.right = right;
    }


}

```



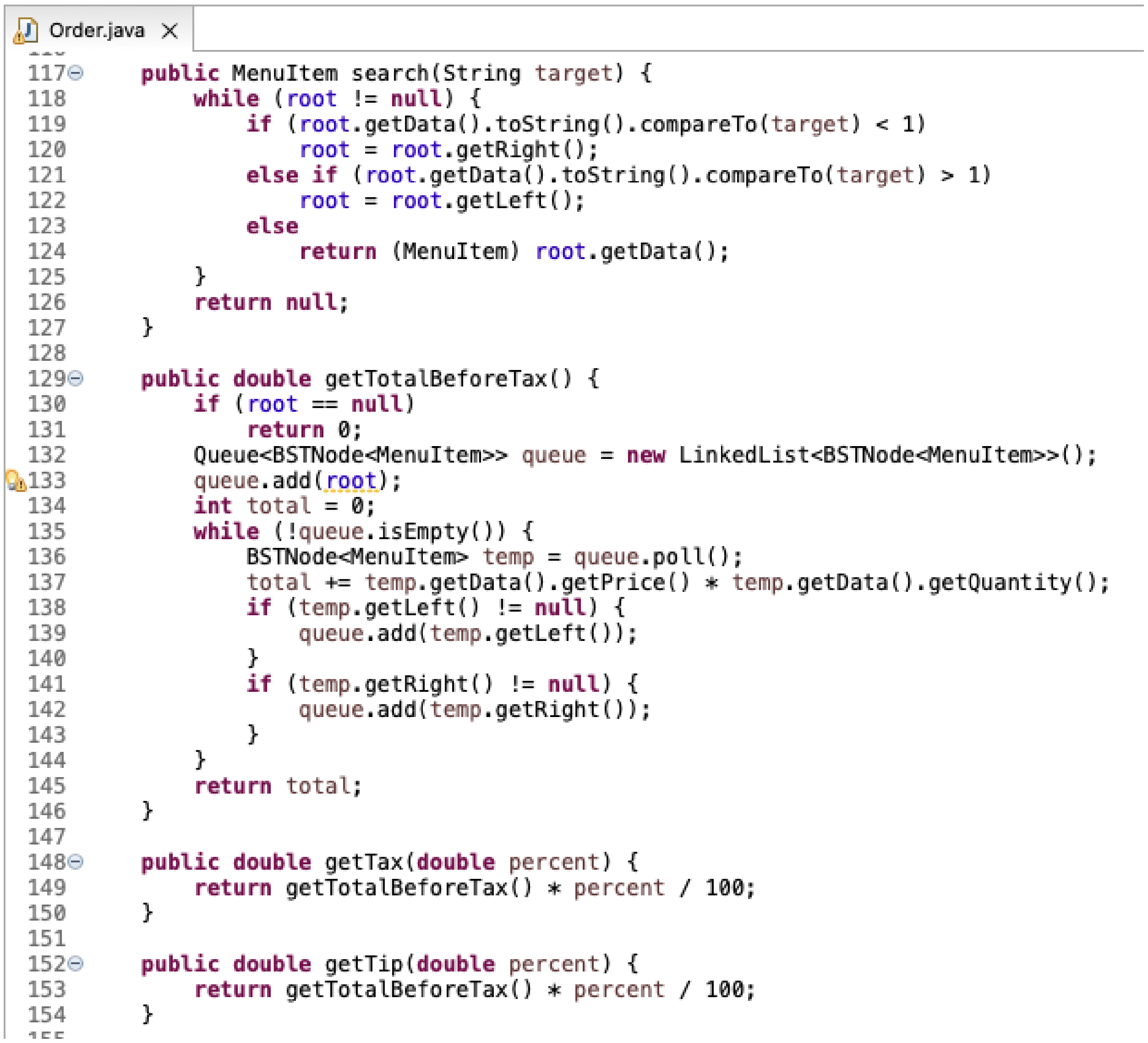
```java
package restaurant;

import java.text.DecimalFormat;

public class MenuItem implements Comparable<MenuItem>{
    private String name;
    private double price;
    private int quantity;

    public MenuItem(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public String toString() {
        DecimalFormat df = new DecimalFormat("0.00");
        return this.getName() + "\t$" + df.format(this.getPrice()) + "\t" + this.getQuantity() + "\t$" + df.format(this.quantity*this.price);
    }

    public boolean equals(MenuItem item) {
        if(this.getName().compareToIgnoreCase(item.getName()) == 0) {
            return true;
        } else {
            return false;
        }
    }

```



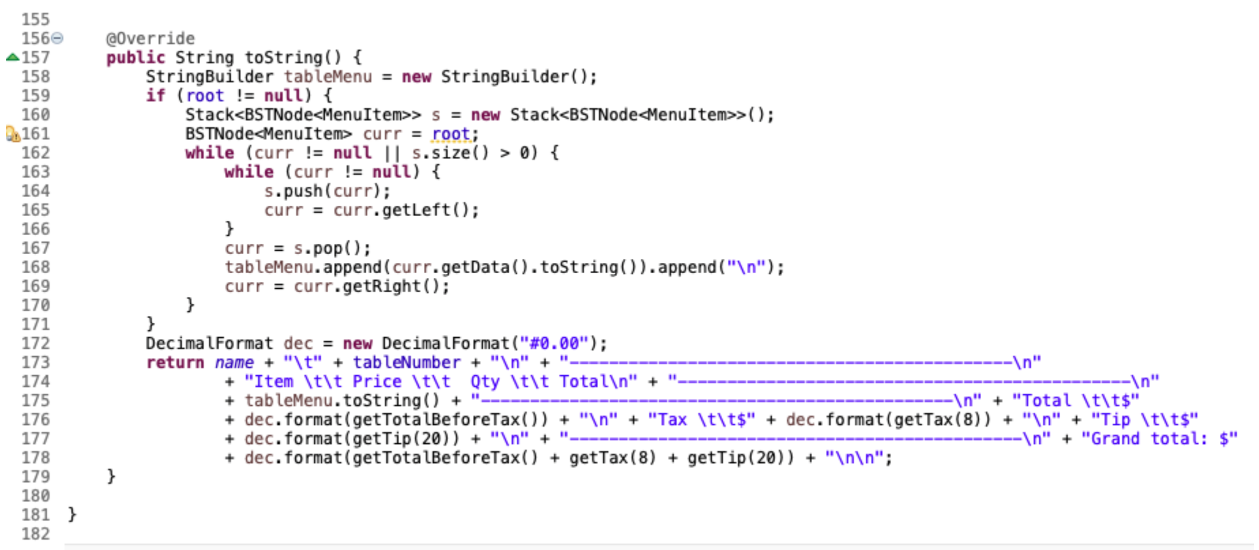
```java

    @Override
    public int compareTo(MenuItem item) {
        if(this.name.compareTo(item.name) == 0) {
            return 0;
        } else if(this.name.compareTo(this.name) > 0) {
            return 1;
        } else {
            return -1;
        }
    }


}
```


Order.java

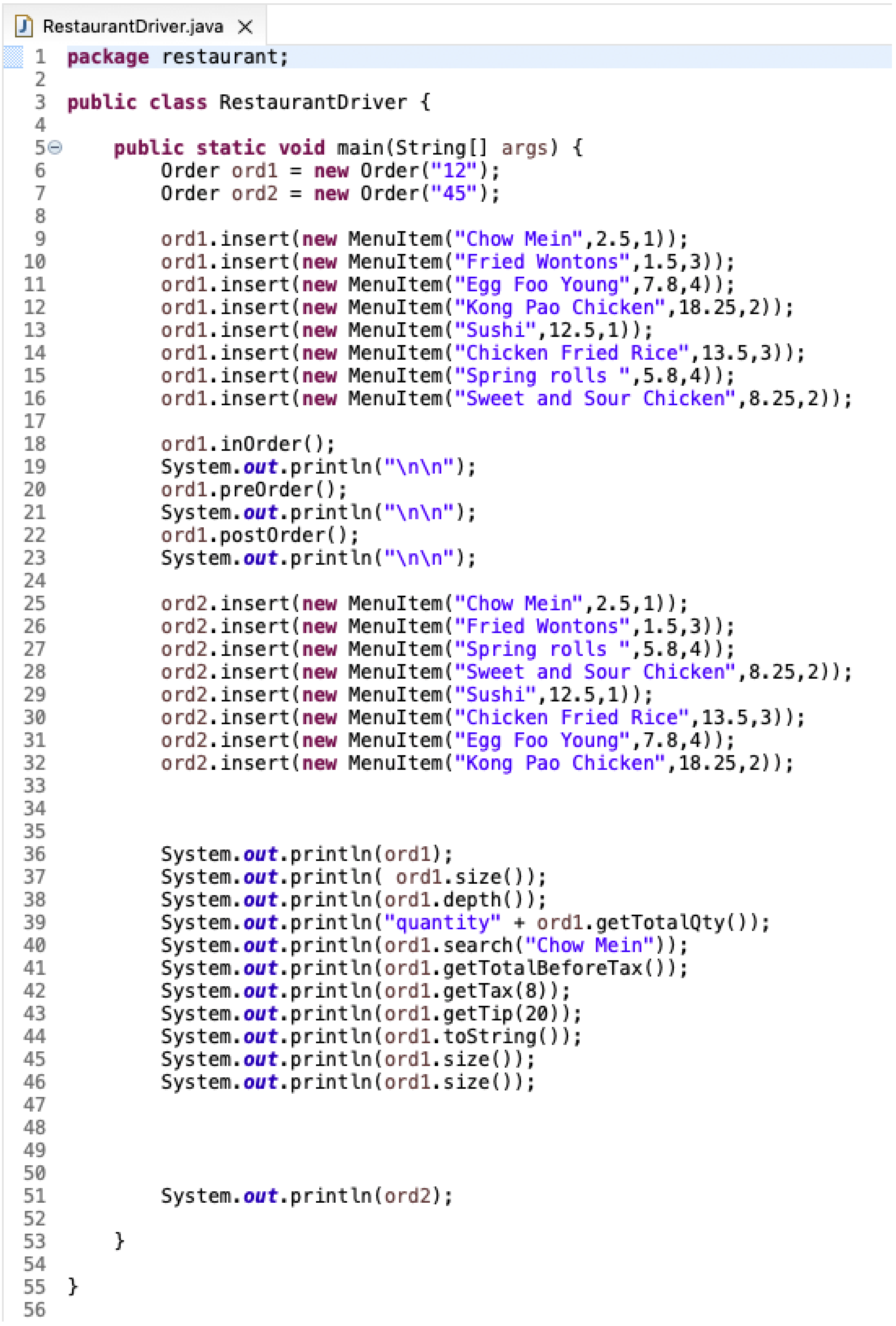
```java
package restaurant;

import java.text.DecimalFormat;

public class Order {
    private BSTNode root;
    private String tableNumber;

    static String name = "Chipotle";

    public Order(String tableNumber) {
        this.tableNumber = tableNumber;
        root = null;
    }

    public void insert(MenuItem menuItem) {
        if (root == null) {
            root = new BSTNode(menuItem, null, null);
        } else {
            insert(root, menuItem);
        }
    }

    public void insert(BSTNode current, MenuItem menuItem) {
        if (menuItem.compareTo((MenuItem) current.getData()) <= 0) {
            if (current.getLeft() == null) {
                current.setLeft(new BSTNode(menuItem, null, null));
            } else {
                insert(current.getLeft(), menuItem);
            }
        } else {
            if (current.getRight() == null) {
                current.setRight(new BSTNode(menuItem, null, null));
            } else {
                insert(current.getRight(), menuItem);
            }
        }
    }

    public void preOrder() {
        preOrder(root);
    }

    public void preOrder(BSTNode node) {
        if (node == null)
            return;
        System.out.print(node.data + "->");
        preOrder(node.left);
        preOrder(node.right);
    }

    public void inOrder() {
        inOrder(root);
    }

```


Order.java

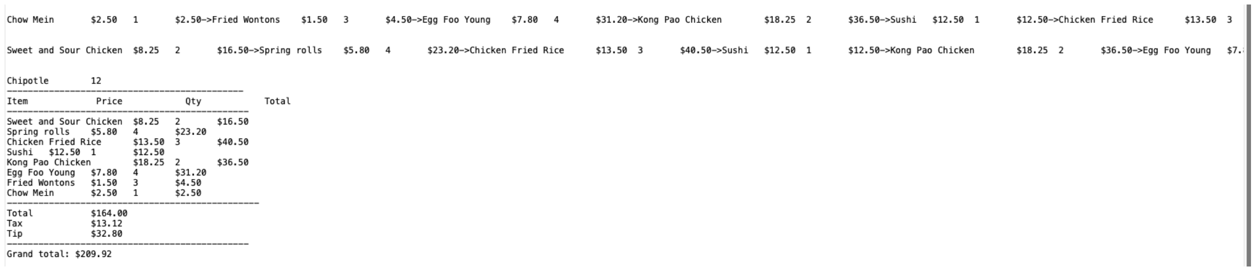
```java
    public MenuItem search(String target) {
        while (root != null) {
            if (root.getData().toString().compareTo(target) < 1)
                root = root.getRight();
            else if (root.getData().toString().compareTo(target) > 1)
                root = root.getLeft();
            else
                return (MenuItem) root.getData();
        }
        return null;
    }

    public double getTotalBeforeTax() {
        if (root == null)
            return 0;
        Queue<BSTNode<MenuItem>> queue = new LinkedList<BSTNode<MenuItem>>();
        queue.add(root);
        int total = 0;
        while (!queue.isEmpty()) {
            BSTNode<MenuItem> temp = queue.poll();
            total += temp.getData().getPrice() * temp.getData().getQuantity();
            if (temp.getLeft() != null) {
                queue.add(temp.getLeft());
            }
            if (temp.getRight() != null) {
                queue.add(temp.getRight());
            }
        }
        return total;
    }

    public double getTax(double percent) {
        return getTotalBeforeTax() * percent / 100;
    }

    public double getTip(double percent) {
        return getTotalBeforeTax() * percent / 100;
    }
```



```java

    @Override
    public String toString() {
        StringBuilder tableMenu = new StringBuilder();
        if (root != null) {
            Stack<BSTNode<MenuItem>> s = new Stack<BSTNode<MenuItem>>();
            BSTNode<MenuItem> curr = root;
            while (curr != null || s.size() > 0) {
                while (curr != null) {
                    s.push(curr);
                    curr = curr.getLeft();
                }
                curr = s.pop();
                tableMenu.append(curr.getData().toString()).append("\n");
                curr = curr.getRight();
            }
        }
        DecimalFormat dec = new DecimalFormat("#0.00");
        return name + "\t" + tableNumber + "\n" + "----------------------------------------------\n"
                + "Item \t\t Price \t\t  Qty \t\t Total\n" + "-----------------------------------------------\n"
                + tableMenu.toString() + "-------------------------------------------------\n" + "Total \t\t$"
                + dec.format(getTotalBeforeTax()) + "\n" + "Tax \t\t$" + dec.format(getTax(8)) + "\n" + "Tip \t\t$"
                + dec.format(getTip(20)) + "\n" + "-----------------------------------------------\n" + "Grand total: $"
                + dec.format(getTotalBeforeTax() + getTax(8) + getTip(20)) + "\n\n";
    }

}

```

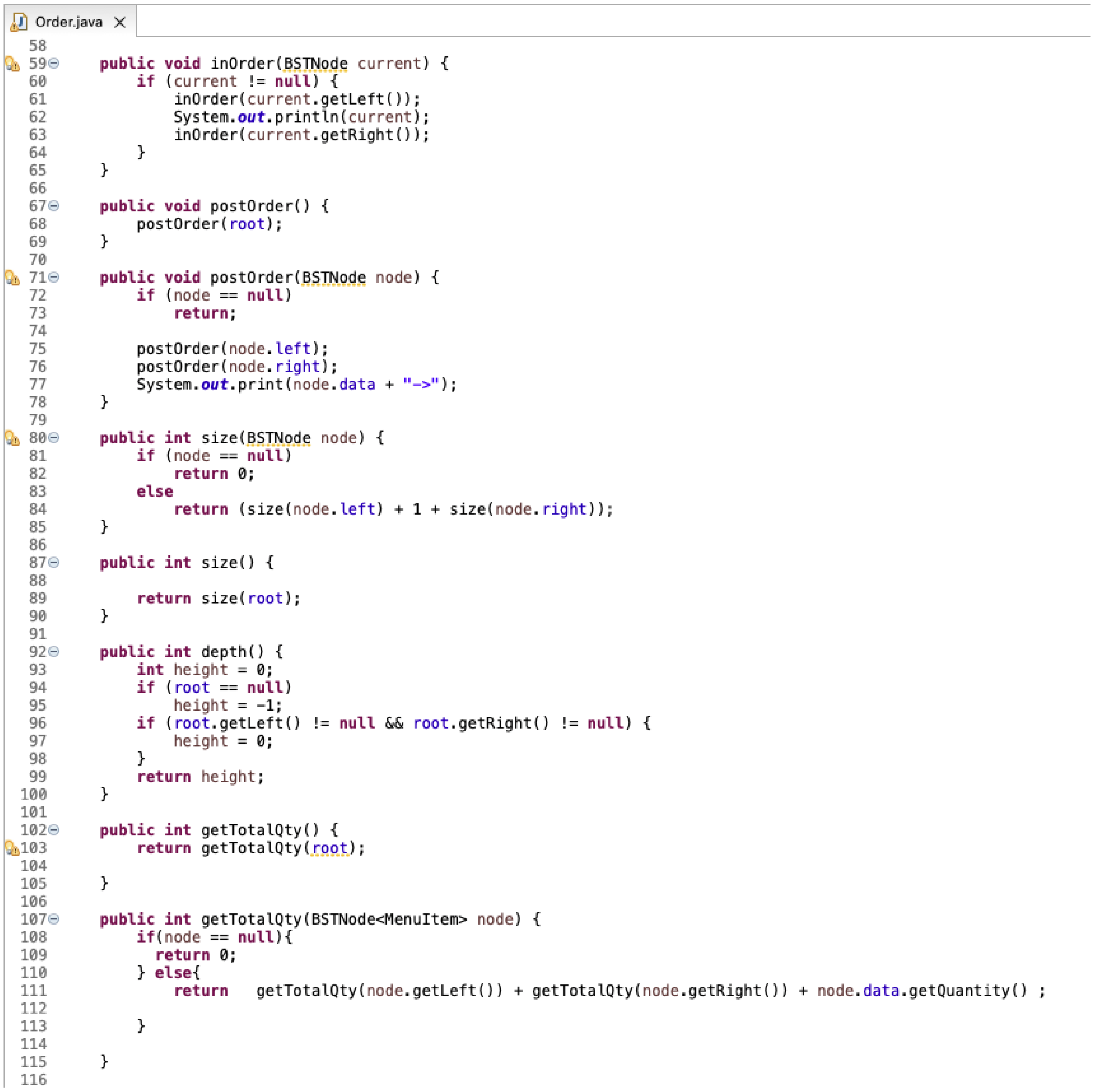

RestaurantDriver.java

```java
package restaurant;

public class RestaurantDriver {

    public static void main(String[] args) {
        Order ord1 = new Order("12");
        Order ord2 = new Order("45");

        ord1.insert(new MenuItem("Chow Mein",2.5,1));
        ord1.insert(new MenuItem("Fried Wontons",1.5,3));
        ord1.insert(new MenuItem("Egg Foo Young",7.8,4));
        ord1.insert(new MenuItem("Kong Pao Chicken",18.25,2));
        ord1.insert(new MenuItem("Sushi",12.5,1));
        ord1.insert(new MenuItem("Chicken Fried Rice",13.5,3));
        ord1.insert(new MenuItem("Spring rolls ",5.8,4));
        ord1.insert(new MenuItem("Sweet and Sour Chicken",8.25,2));

        ord1.inOrder();
        System.out.println("\n\n");
        ord1.preOrder();
        System.out.println("\n\n");
        ord1.postOrder();
        System.out.println("\n\n");

        ord2.insert(new MenuItem("Chow Mein",2.5,1));
        ord2.insert(new MenuItem("Fried Wontons",1.5,3));
        ord2.insert(new MenuItem("Spring rolls ",5.8,4));
        ord2.insert(new MenuItem("Sweet and Sour Chicken",8.25,2));
        ord2.insert(new MenuItem("Sushi",12.5,1));
        ord2.insert(new MenuItem("Chicken Fried Rice",13.5,3));
        ord2.insert(new MenuItem("Egg Foo Young",7.8,4));
        ord2.insert(new MenuItem("Kong Pao Chicken",18.25,2));



        System.out.println(ord1);
        System.out.println( ord1.size());
        System.out.println(ord1.depth());
        System.out.println("quantity" + ord1.getTotalQty());
        System.out.println(ord1.search("Chow Mein"));
        System.out.println(ord1.getTotalBeforeTax());
        System.out.println(ord1.getTax(8));
        System.out.println(ord1.getTip(20));
        System.out.println(ord1.toString());
        System.out.println(ord1.size());
        System.out.println(ord1.size());




        System.out.println(ord2);

    }

}
```



```
Chow Mein       $2.50   1       $2.50->Fried Wontons    $1.50   3       $4.50->Egg Foo Young    $7.80   4       $31.20->Kong Pao Chicken        $18.25  2       $36.50->Sushi   $12.50  1       $12.50->Chicken Fried Rice      $13.50  3

Sweet and Sour Chicken  $8.25   2       $16.50->Spring rolls    $5.80   4       $23.20->Chicken Fried Rice      $13.50  3       $40.50->Sushi   $12.50  1       $12.50->Kong Pao Chicken        $18.25  2       $36.50->Egg Foo Young   $7.

Chipotle        12
---------------------------------------------
Item             Price            Qty            Total
----------------------------------------------
Sweet and Sour Chicken  $8.25   2       $16.50
Spring rolls    $5.80   4       $23.20
Chicken Fried Rice      $13.50  3       $40.50
Sushi   $12.50  1       $12.50
Kong Pao Chicken        $18.25  2       $36.50
Egg Foo Young   $7.80   4       $31.20
Fried Wontons   $1.50   3       $4.50
Chow Mein       $2.50   1       $2.50
------------------------------------------------
Total           $164.00
Tax             $13.12
Tip             $32.80
----------------------------------------------
Grand total: $209.92
```

```
8
0
quantity20
null
0.0
0.0
0.0
Chipotle        12
---------------------------------------------
Item             Price            Qty            Total
----------------------------------------------
------------------------------------------------
Total           $0.00
Tax             $0.00
Tip             $0.00
----------------------------------------------
Grand total: $0.00


0
0
Chipotle        45
---------------------------------------------
Item             Price            Qty            Total
----------------------------------------------
Kong Pao Chicken        $18.25  2       $36.50
Egg Foo Young   $7.80   4       $31.20
Chicken Fried Rice      $13.50  3       $40.50
Sushi   $12.50  1       $12.50
Sweet and Sour Chicken  $8.25   2       $16.50
Spring rolls    $5.80   4       $23.20
Fried Wontons   $1.50   3       $4.50
Chow Mein       $2.50   1       $2.50
------------------------------------------------
Total           $164.00
Tax             $13.12
Tip             $32.80
----------------------------------------------
Grand total: $209.92
```



```java

    public void inOrder(BSTNode current) {
        if (current != null) {
            inOrder(current.getLeft());
            System.out.println(current);
            inOrder(current.getRight());
        }
    }

    public void postOrder() {
        postOrder(root);
    }

    public void postOrder(BSTNode node) {
        if (node == null)
            return;

        postOrder(node.left);
        postOrder(node.right);
        System.out.print(node.data + "->");
    }

    public int size(BSTNode node) {
        if (node == null)
            return 0;
        else
            return (size(node.left) + 1 + size(node.right));
    }

    public int size() {

        return size(root);
    }

    public int depth() {
        int height = 0;
        if (root == null)
            height = -1;
        if (root.getLeft() != null && root.getRight() != null) {
            height = 0;
        }
        return height;
    }

    public int getTotalQty() {
        return getTotalQty(root);

    }

    public int getTotalQty(BSTNode<MenuItem> node) {
        if(node == null){
          return 0;
        } else{
            return   getTotalQty(node.getLeft()) + getTotalQty(node.getRight()) + node.data.getQuantity() ;

        }

    }

```