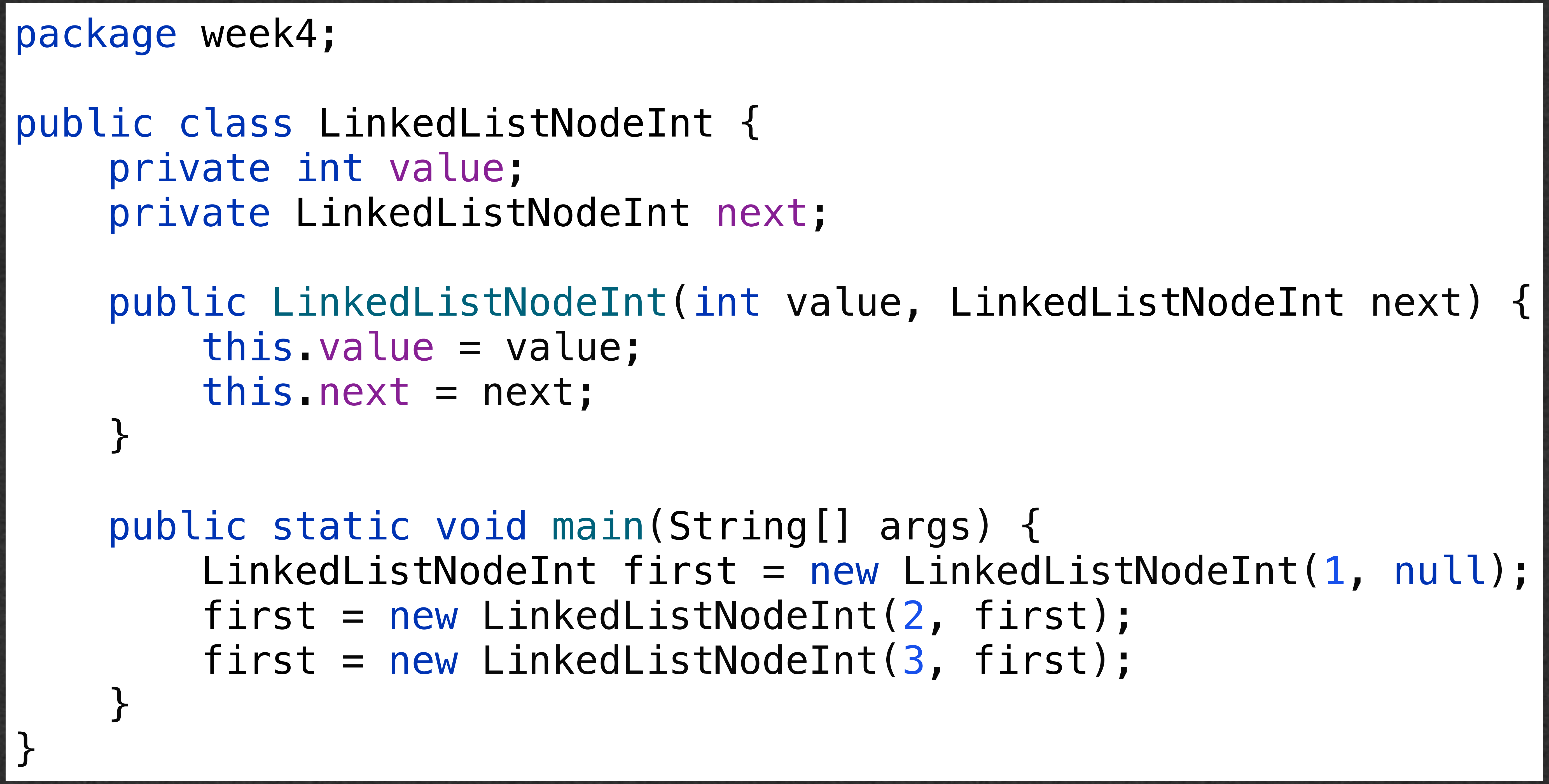
Linked List

Generics

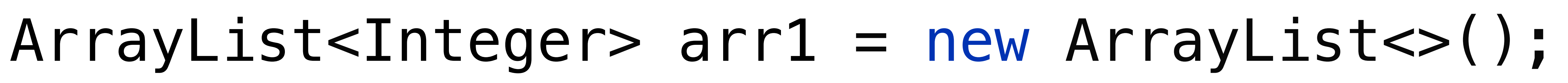
Linked List - Generics



```
package week4;

public class LinkedListNodeInt {
    private int value;
    private LinkedListNodeInt next;

    public LinkedListNodeInt(int value, LinkedListNodeInt next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNodeInt first = new LinkedListNodeInt(1, null);
        first = new LinkedListNodeInt(2, first);
        first = new LinkedListNodeInt(3, first);
    }
}
```

- Last time: We saw this Linked List
- Cool.. but this can only store ints
- What if we want to store anything else?

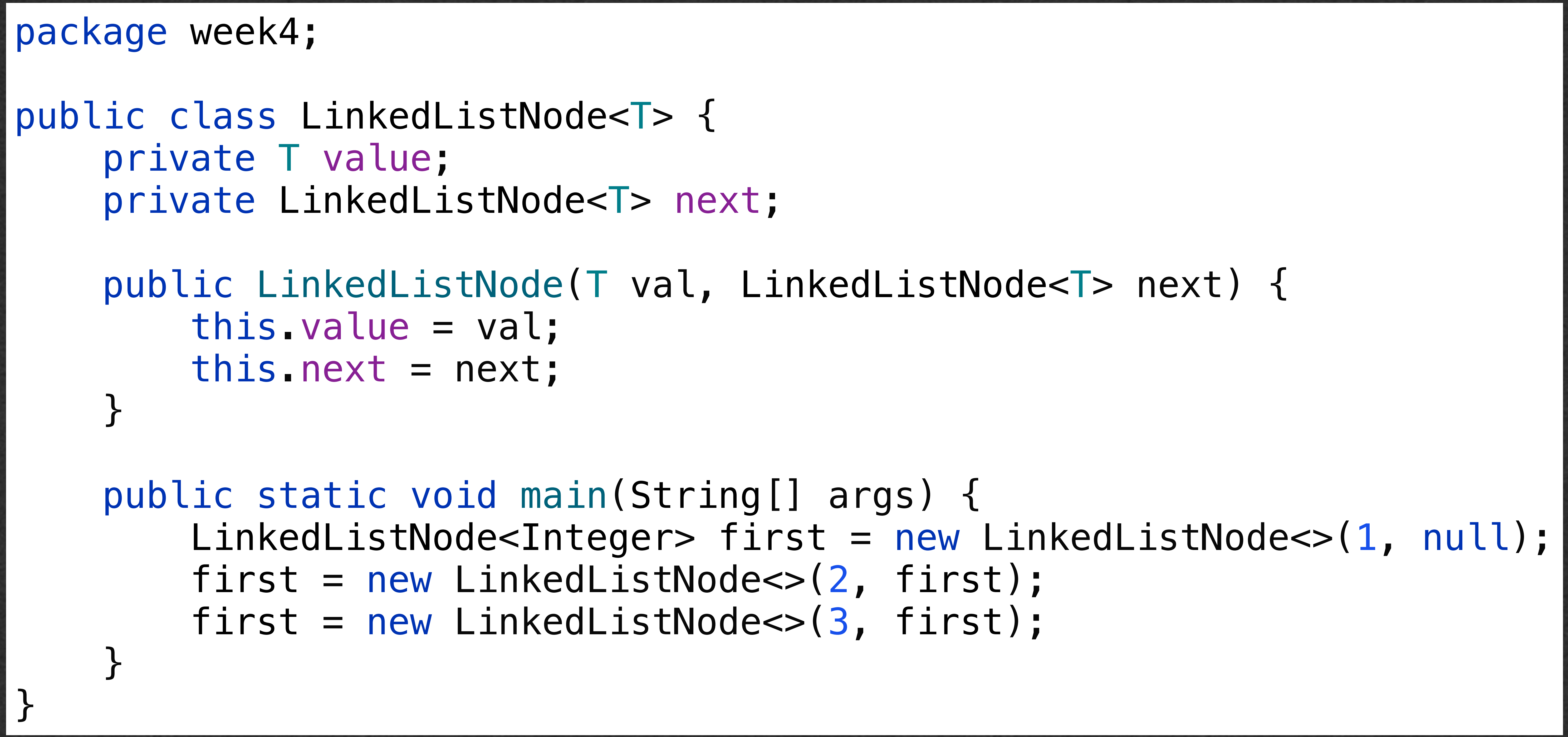
Linked List - Generics



```
ArrayList<Integer> arr1 = new ArrayList<>();
```

- When we create an ArrayList, we give it a type
- The ArrayList can store values of that type
- We want the same functionality in our linked list

Linked List - Generics

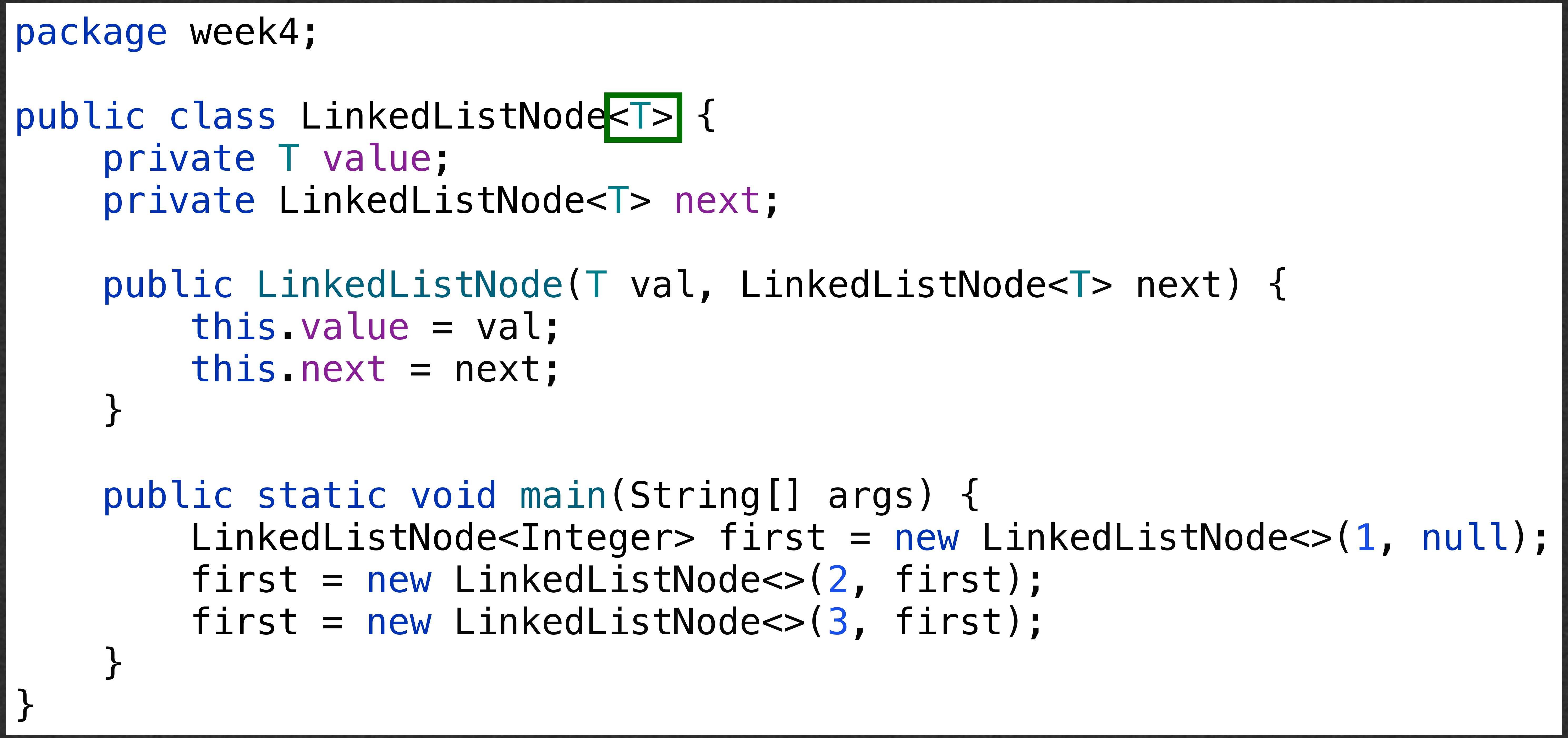


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<Integer> first = new LinkedListNode<>(1, null);
        first = new LinkedListNode<>(2, first);
        first = new LinkedListNode<>(3, first);
    }
}
```

Generics

- Replace every instance of a type with a variable
- The type is set when we create a Linked List

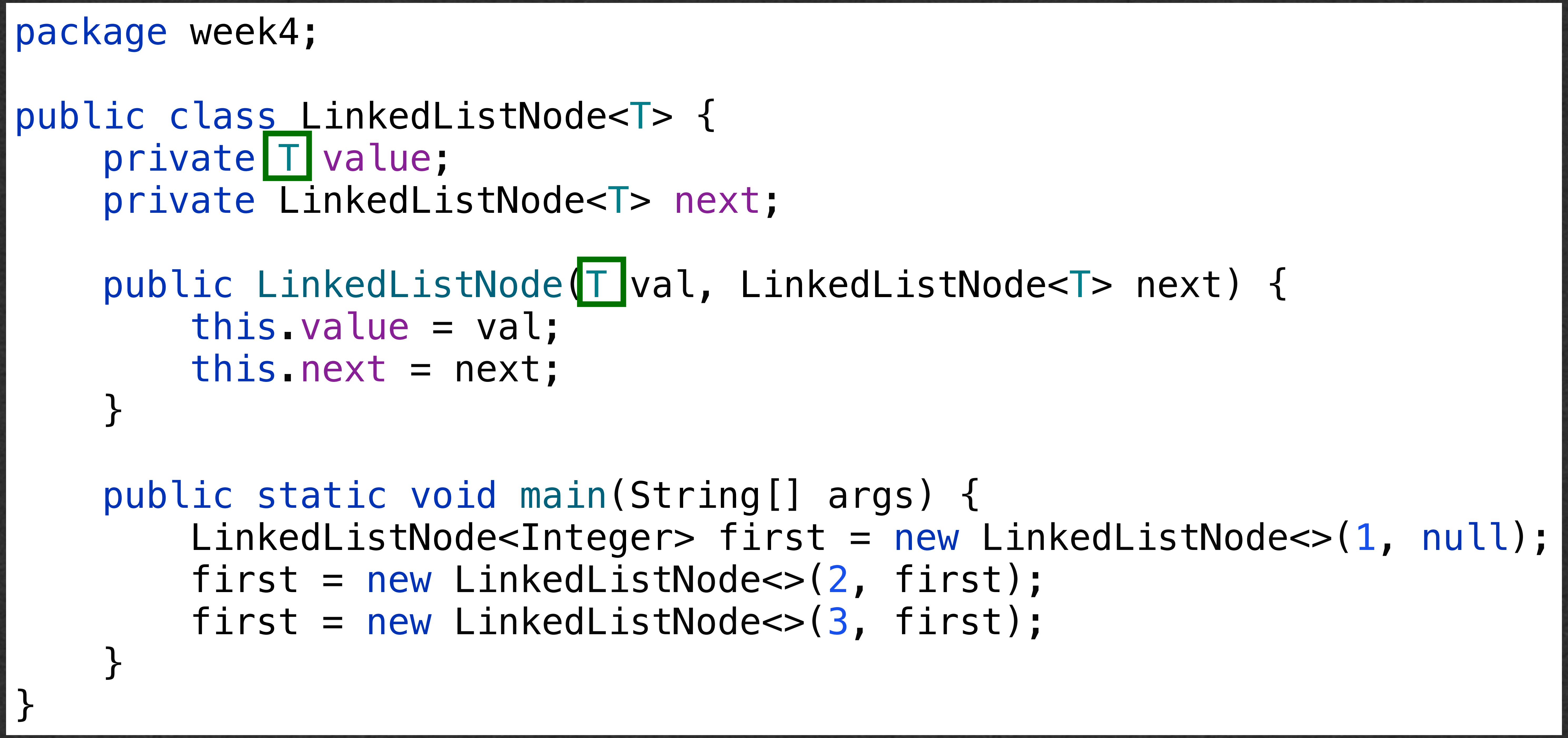
Linked List - Generics



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<Integer> first = new LinkedListNode<>(1, null);
        first = new LinkedListNode<>(2, first);
        first = new LinkedListNode<>(3, first);
    }
}
```

- After the name of the class
    - Add a generic variable in < >
    - This variable is named T

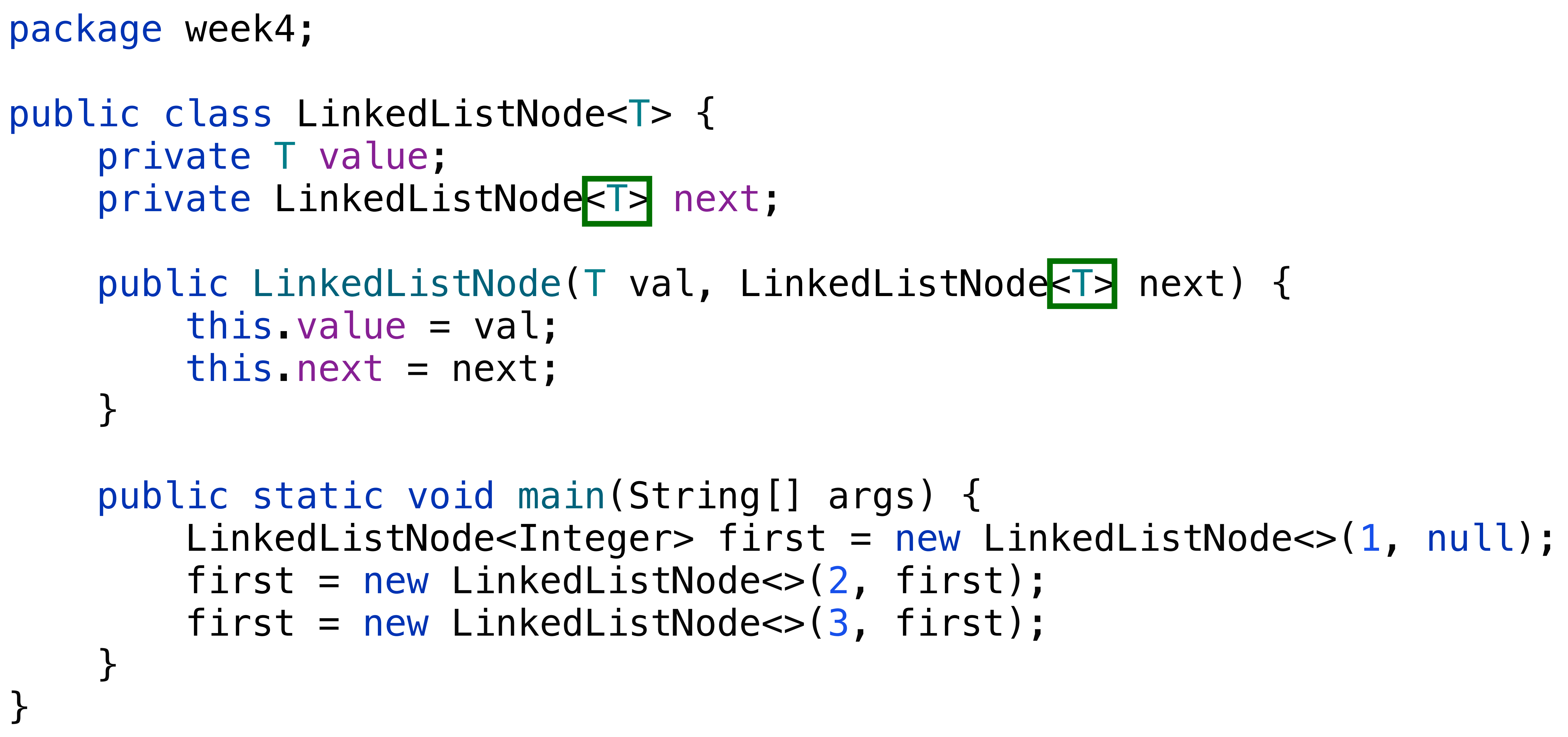
Linked List - Generics



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<Integer> first = new LinkedListNode<>(1, null);
        first = new LinkedListNode<>(2, first);
        first = new LinkedListNode<>(3, first);
    }
}
```

- Whenever you need the type of the List, use T
  - An instance variable of type T
  - A constructor parameter of type T

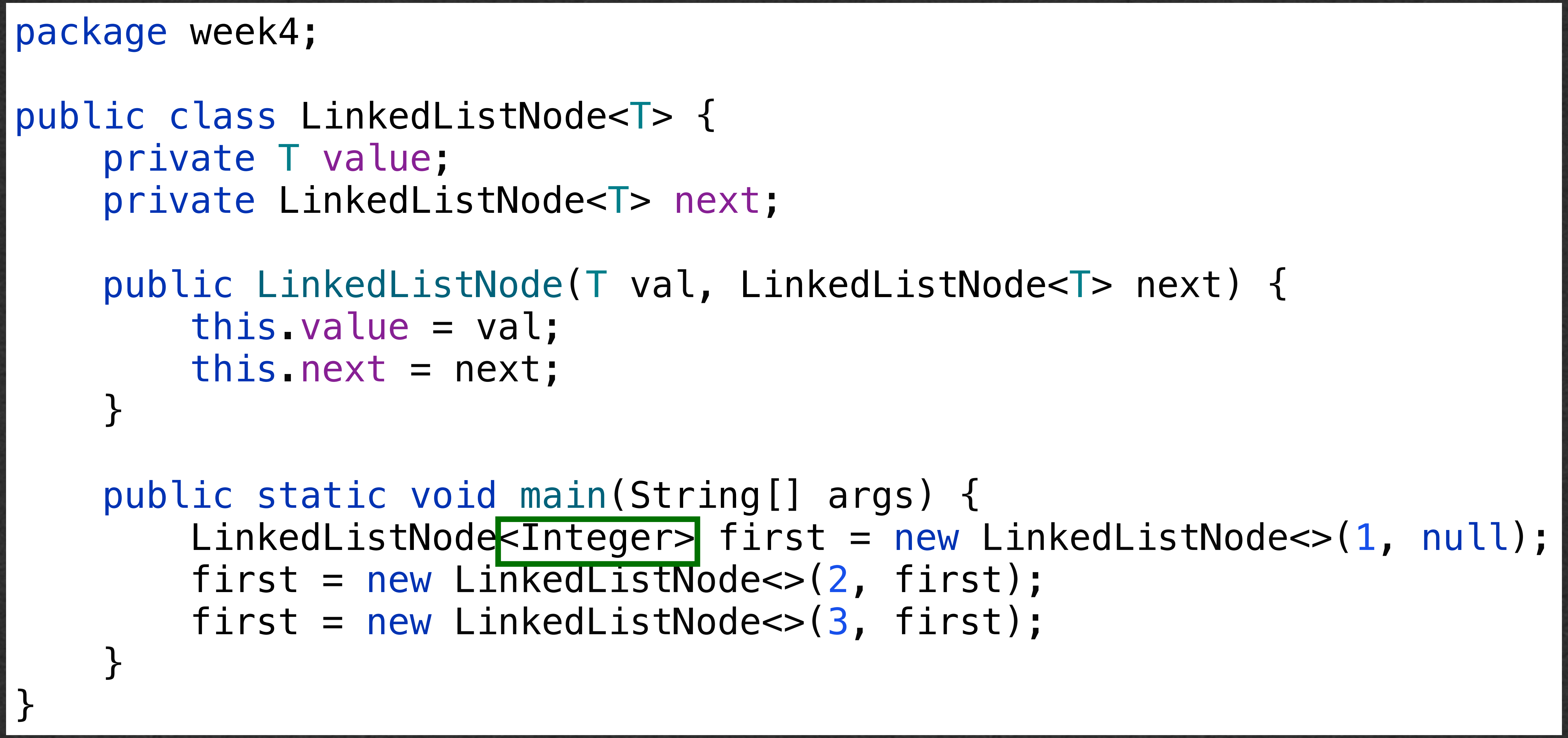
Linked List - Generics



```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<Integer> first = new LinkedListNode<>(1, null);
        first = new LinkedListNode<>(2, first);
        first = new LinkedListNode<>(3, first);
    }
}
```

- Variables of type LinkedListNode need to specify the generic type
    - A node of type T should have a reference to a node of type T
    - All nodes in a Linked List have the same type T

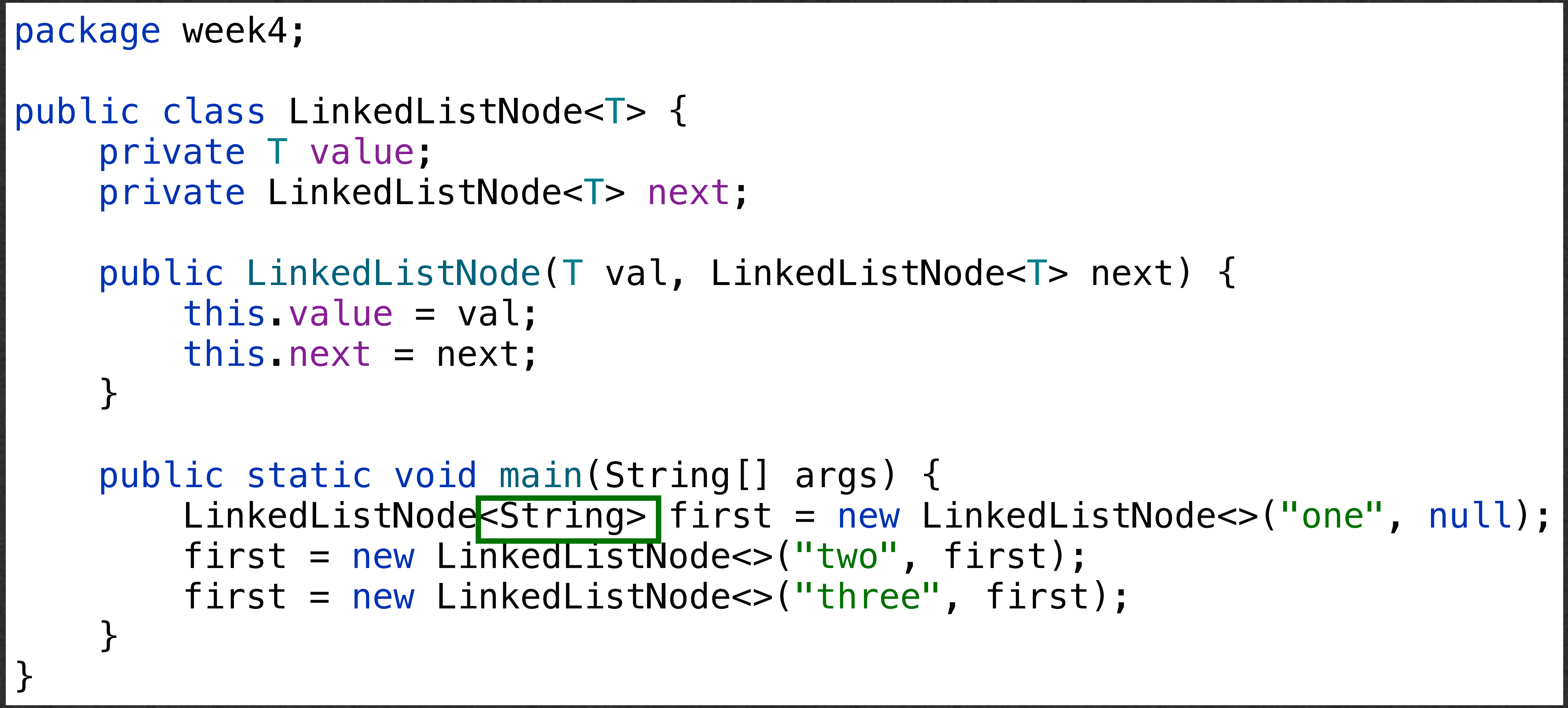
Linked List - Generics



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<Integer> first = new LinkedListNode<>(1, null);
        first = new LinkedListNode<>(2, first);
        first = new LinkedListNode<>(3, first);
    }
}
```

- When we create a new Linked List, we set the value of T to the type for that list
- Here, we set T to Integer
- For this List, all T's will effectively be Integer

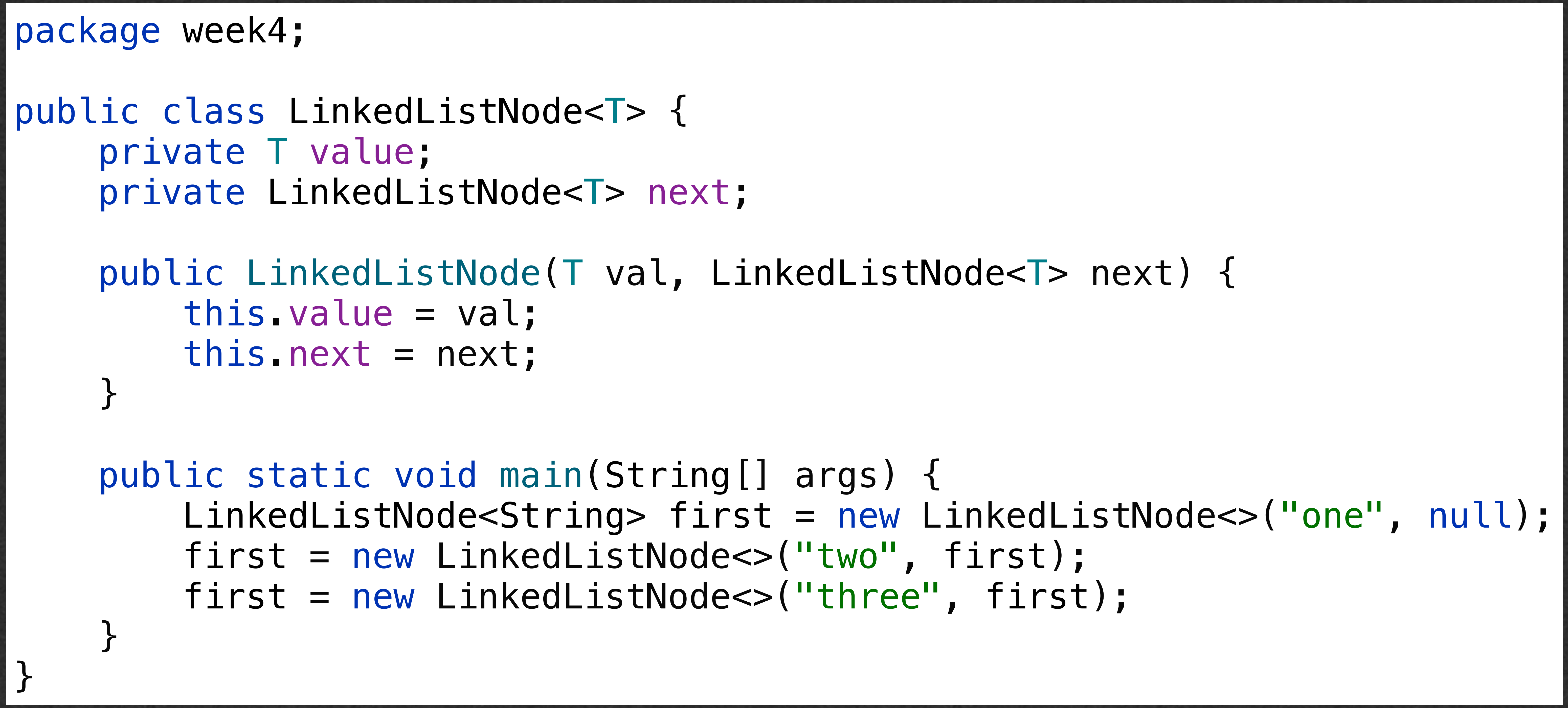
Linked List - Generics



```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<String> first = new LinkedListNode<>("one", null);
        first = new LinkedListNode<>("two", first);
        first = new LinkedListNode<>("three", first);
    }
}
```

- We can set T to any java type (Except primitives)
- For this Linked List, all T's are effectively String

Linked List - Generics



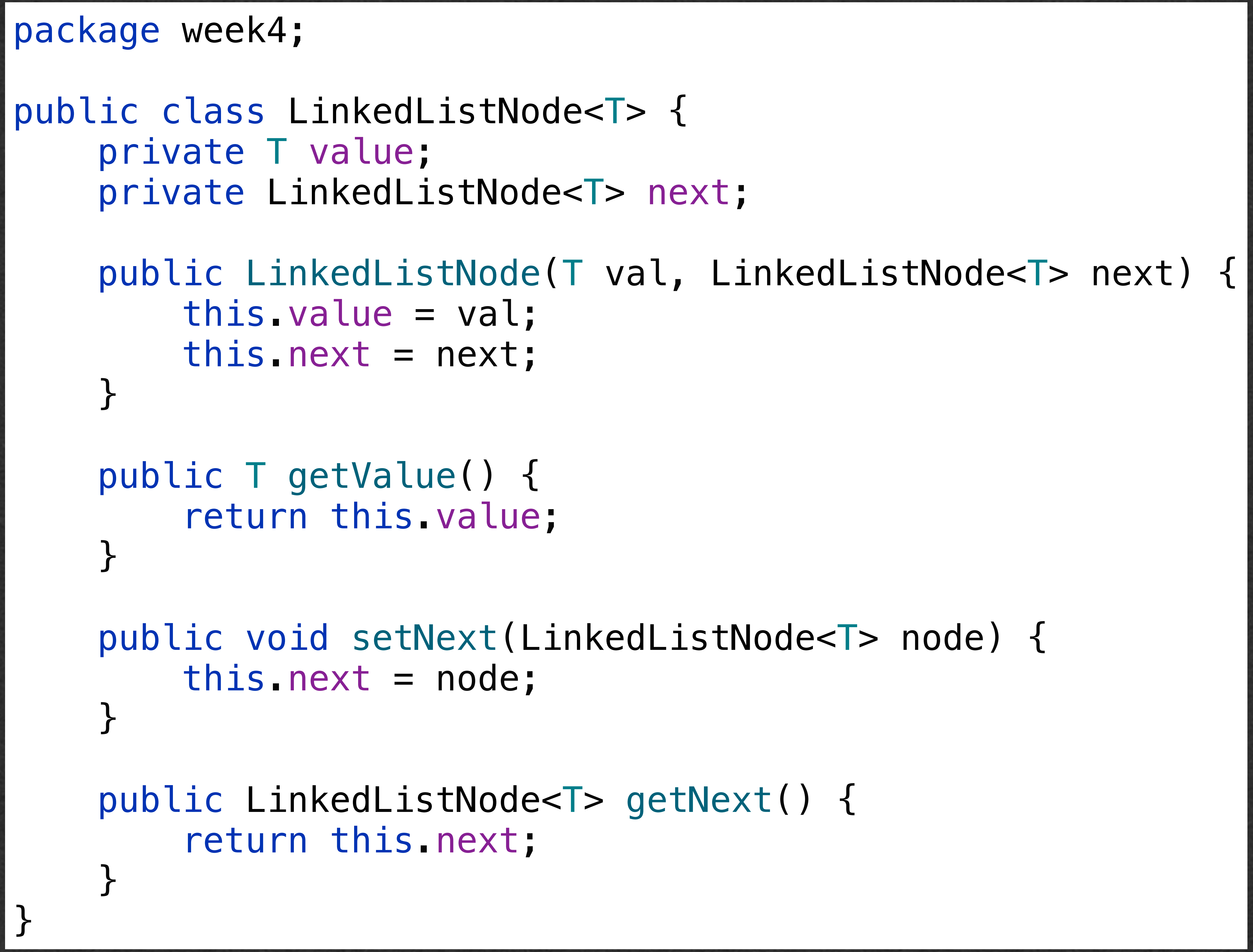
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNode<String> first = new LinkedListNode<>("one", null);
        first = new LinkedListNode<>("two", first);
        first = new LinkedListNode<>("three", first);
    }
}
```

- Using generics allows us to write one class that can store values of any type
- Write Linked List code once
  - Create Linked Lists that can store any type

Algorithms

Linked List - getters/setters

- Let's start adding more functionality to our Linked List
- Start with getters and setters
- When writing getters/setters
  - Only write the methods you'll need
  - We won't change the value of a node, so no setValue method

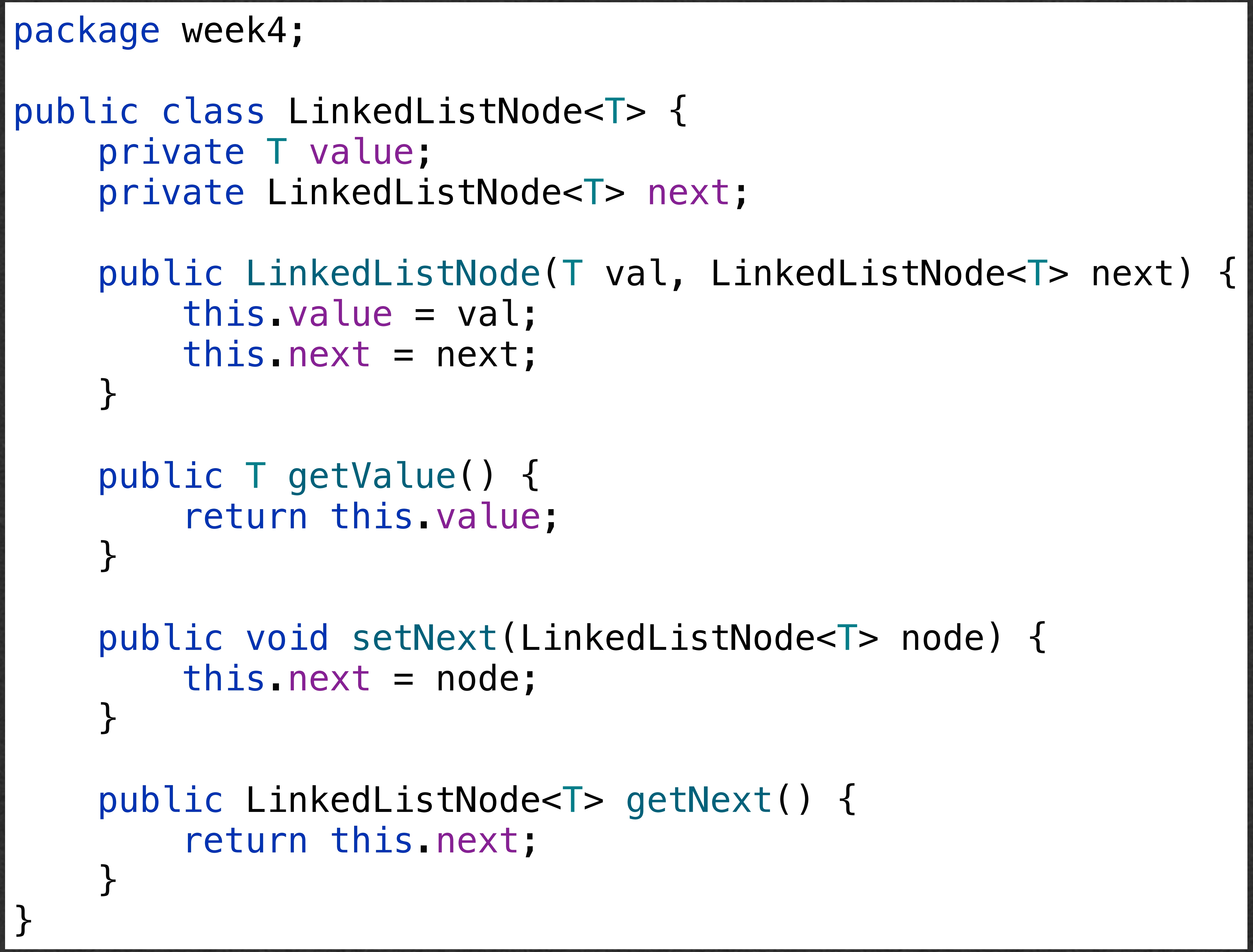


```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public T getValue() {
        return this.value;
    }

    public void setNext(LinkedListNode<T> node) {
        this.next = node;
    }

    public LinkedListNode<T> getNext() {
        return this.next;
    }
}
```

Linked List - Algorithms

- This gives us the basic structure of a Linked List

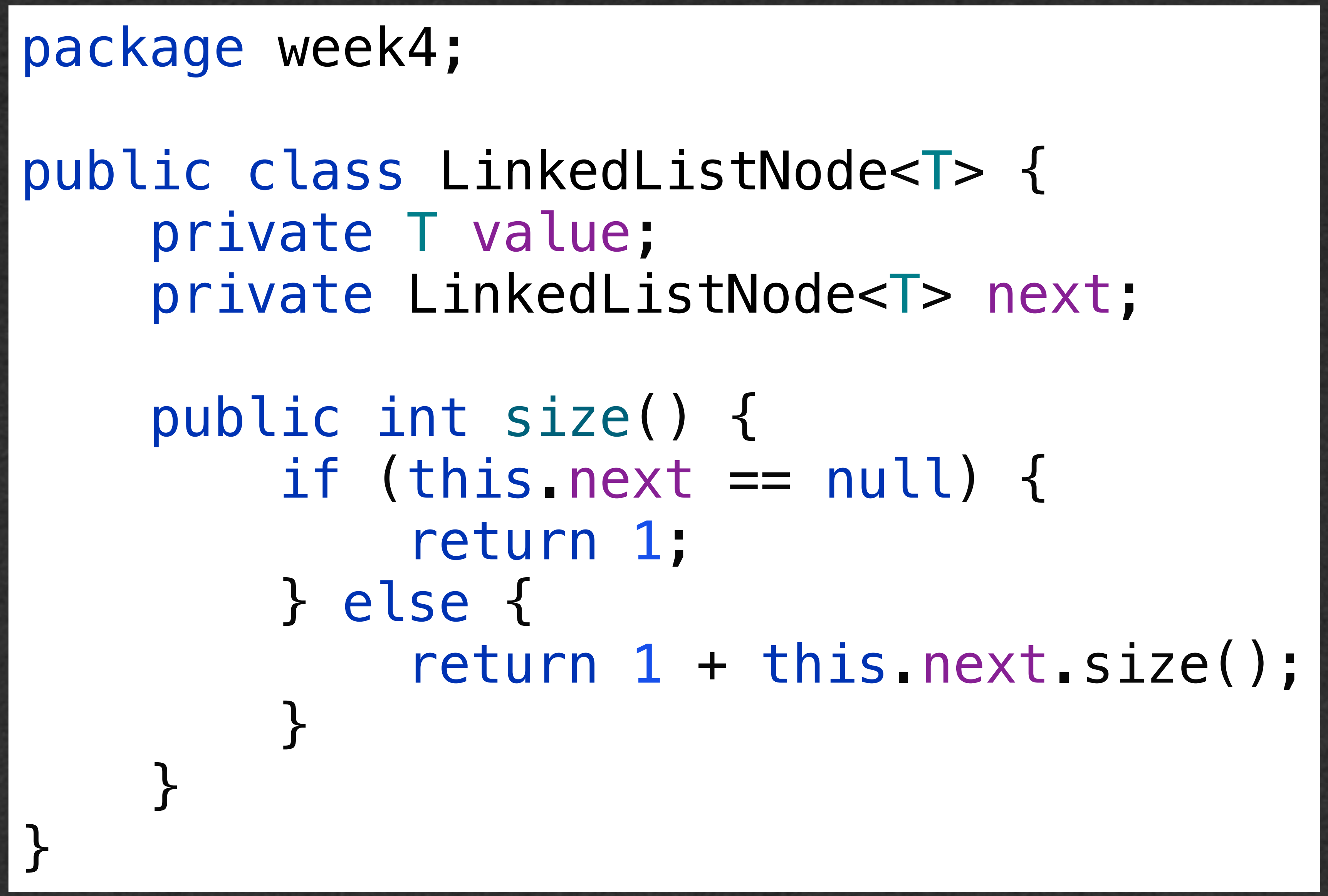
- We'll implements these algorithms as methods in this class
  - size - return the size of the list
  - append - add an element to the end of the list
  - find - return the first node containing a specific value
  - min - find the min value in a list of doubles



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode(T val, LinkedListNode<T> next) {
        this.value = val;
        this.next = next;
    }

    public T getValue() {
        return this.value;
    }

    public void setNext(LinkedListNode<T> node) {
        this.next = node;
    }

    public LinkedListNode<T> getNext() {
        return this.next;
    }
}
```

Linked List - Size

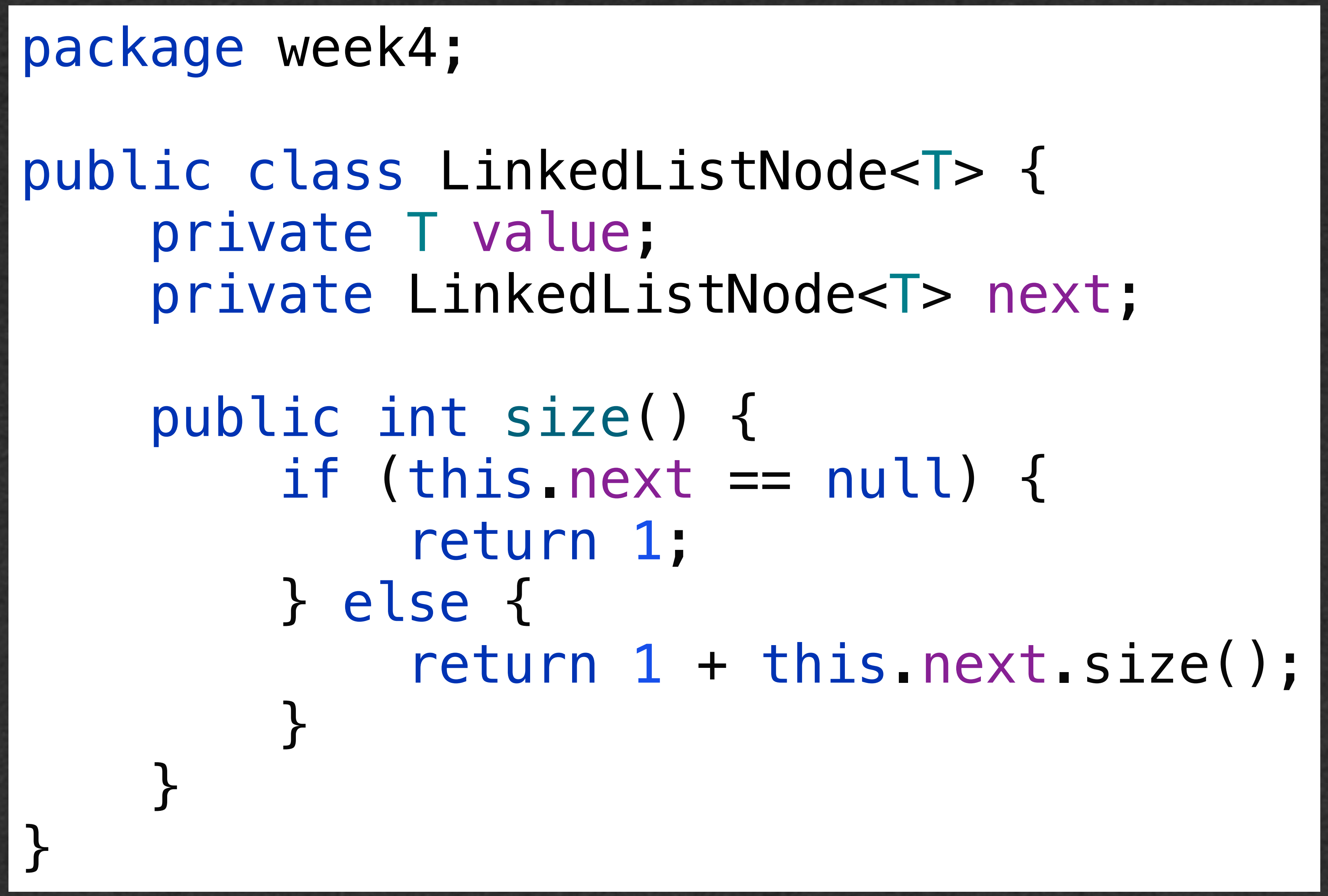
- Navigate through the entire list until the next reference is null
    - Count the number of nodes visited



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```

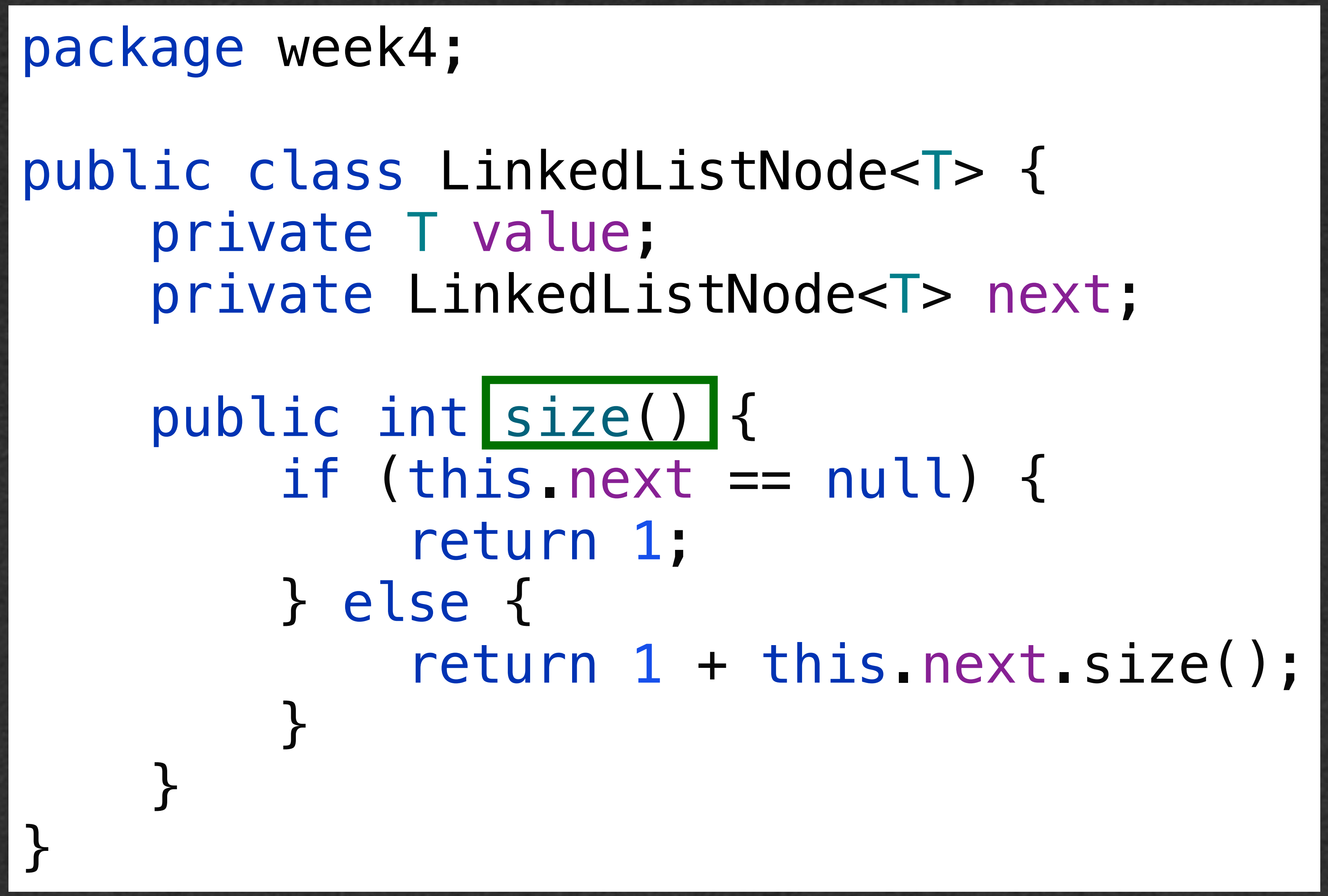
Linked List - Size

- Each node "asks" the next node how many more nodes there are
  - Adds one to the answer and returns
- Last node returns 1

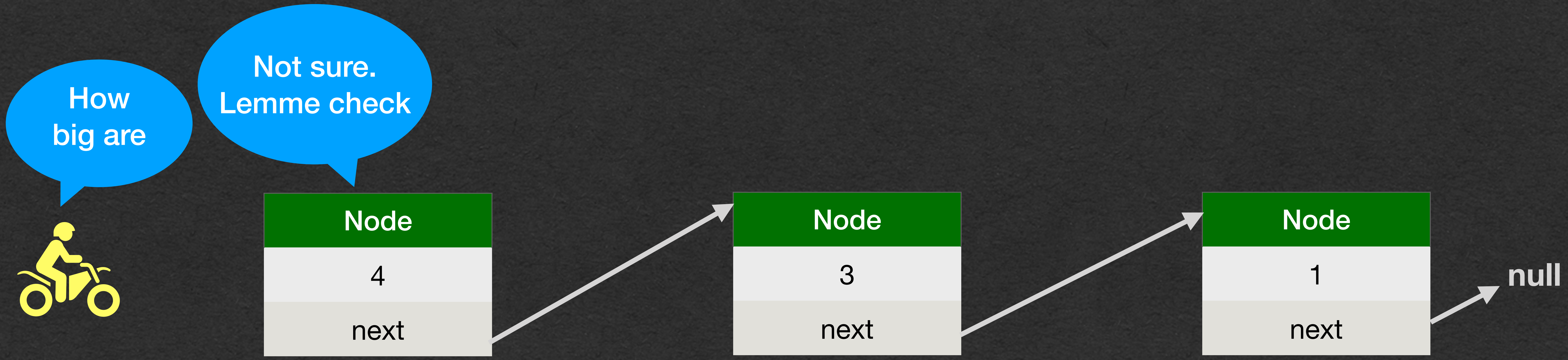


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```

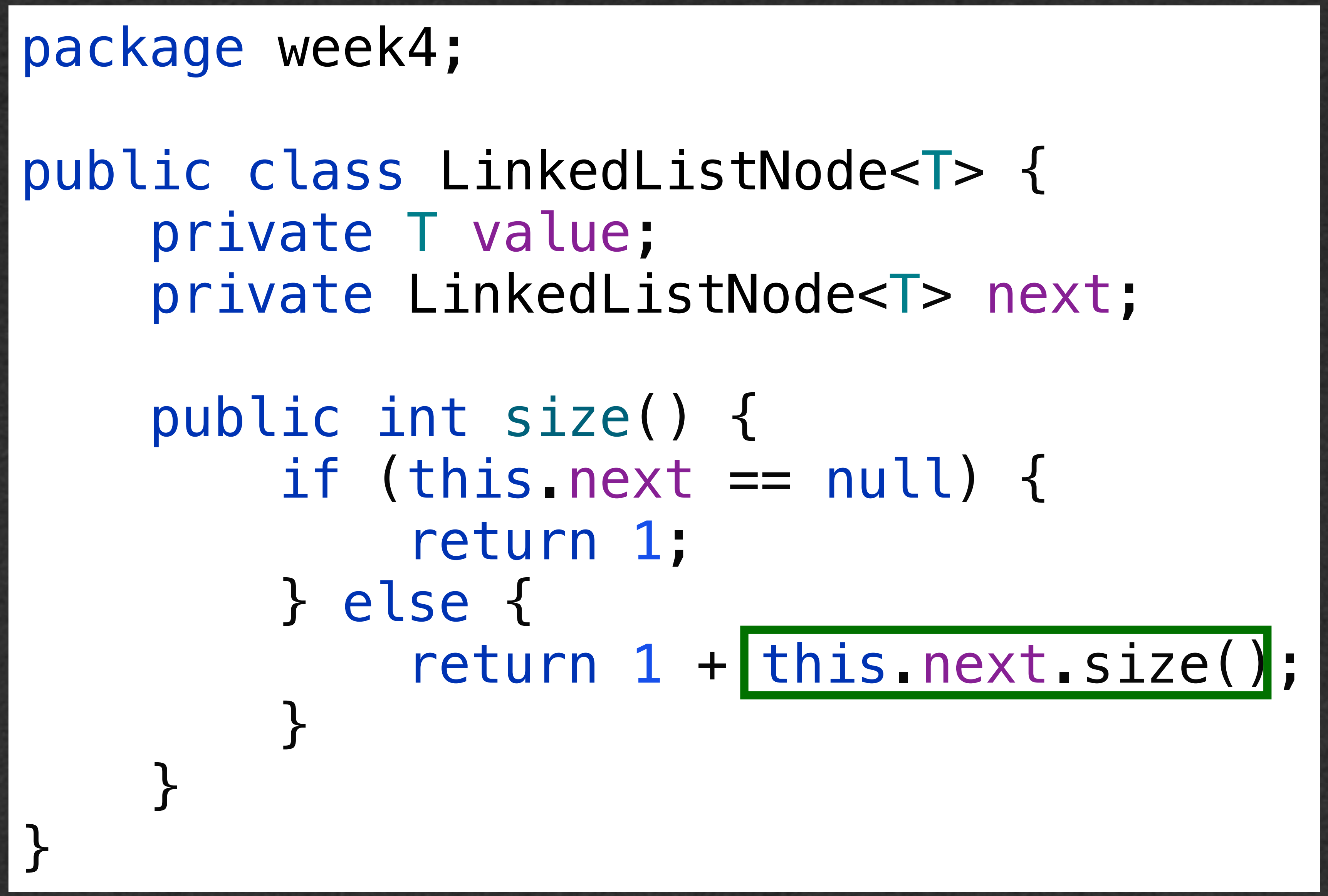
Linked List - Size



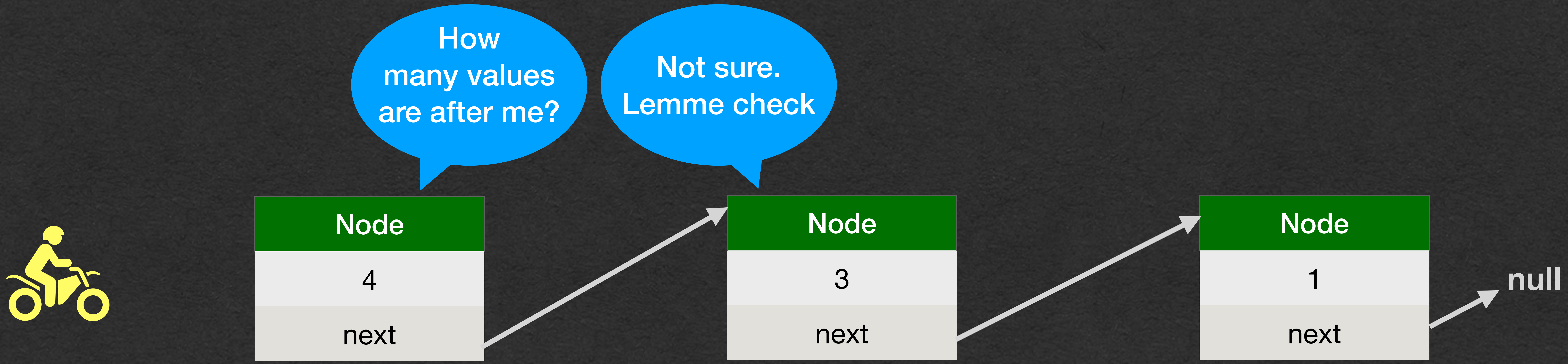
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



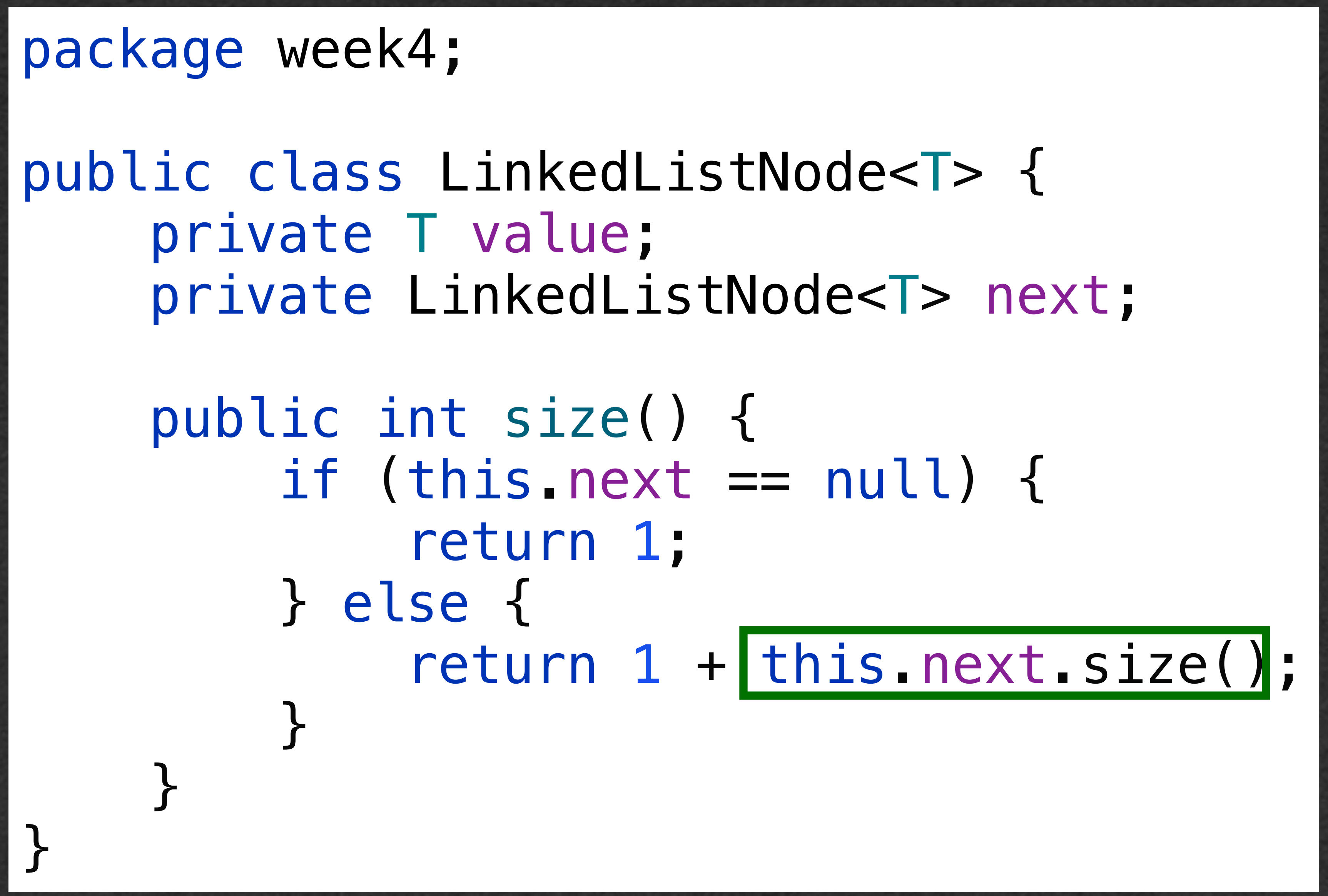
Linked List - Size



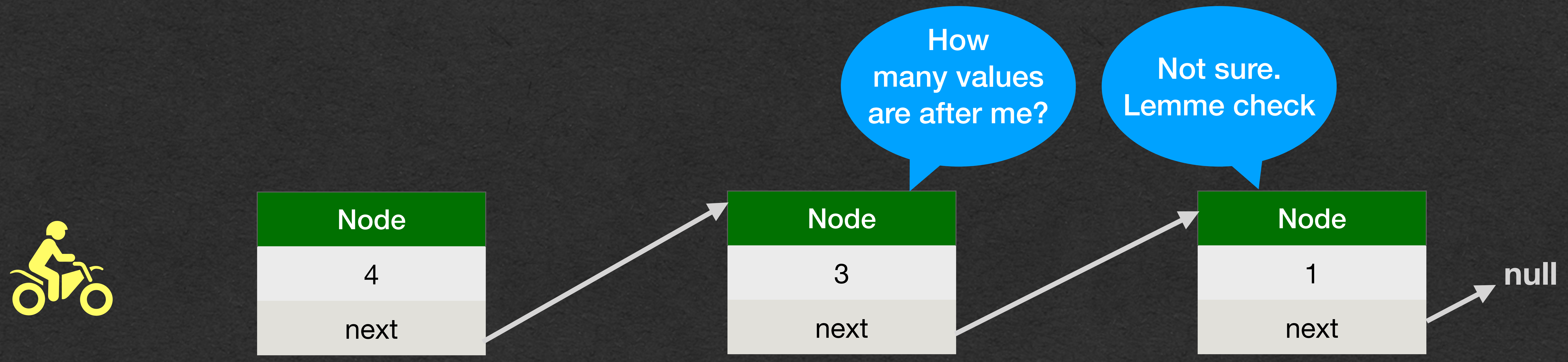
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



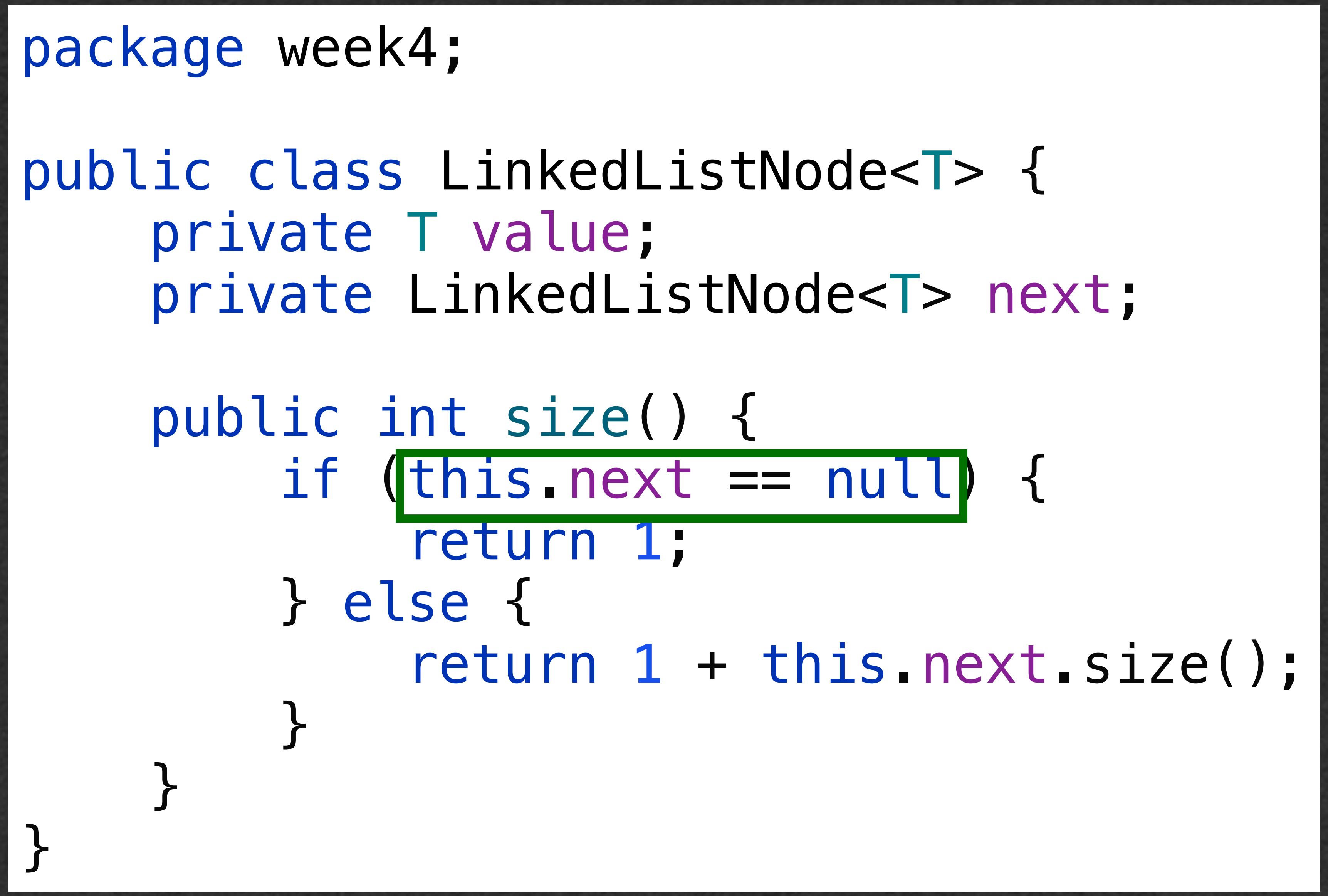
Linked List - Size



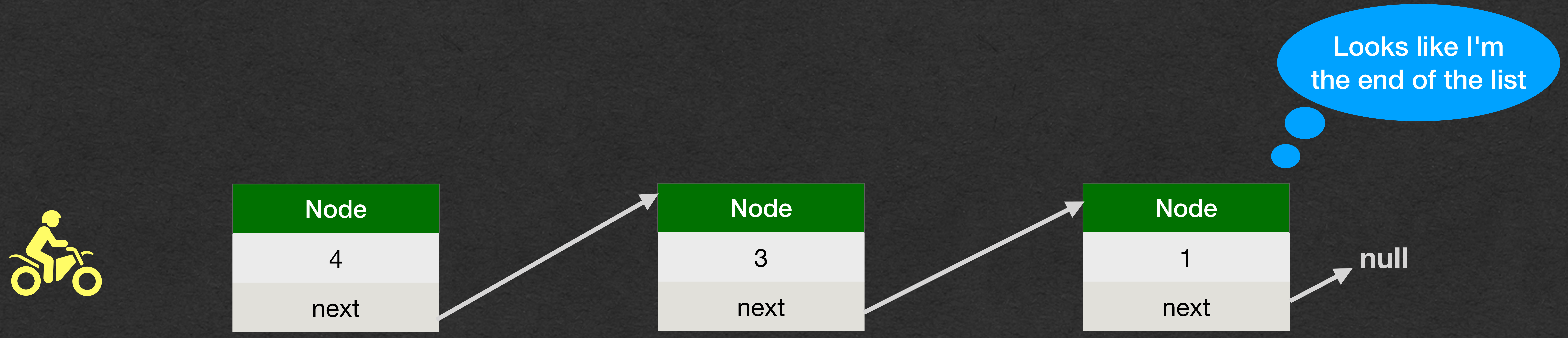
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



Linked List - Size



```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



Looks like I'm the end of the list

Node

4

next

Node

3

next

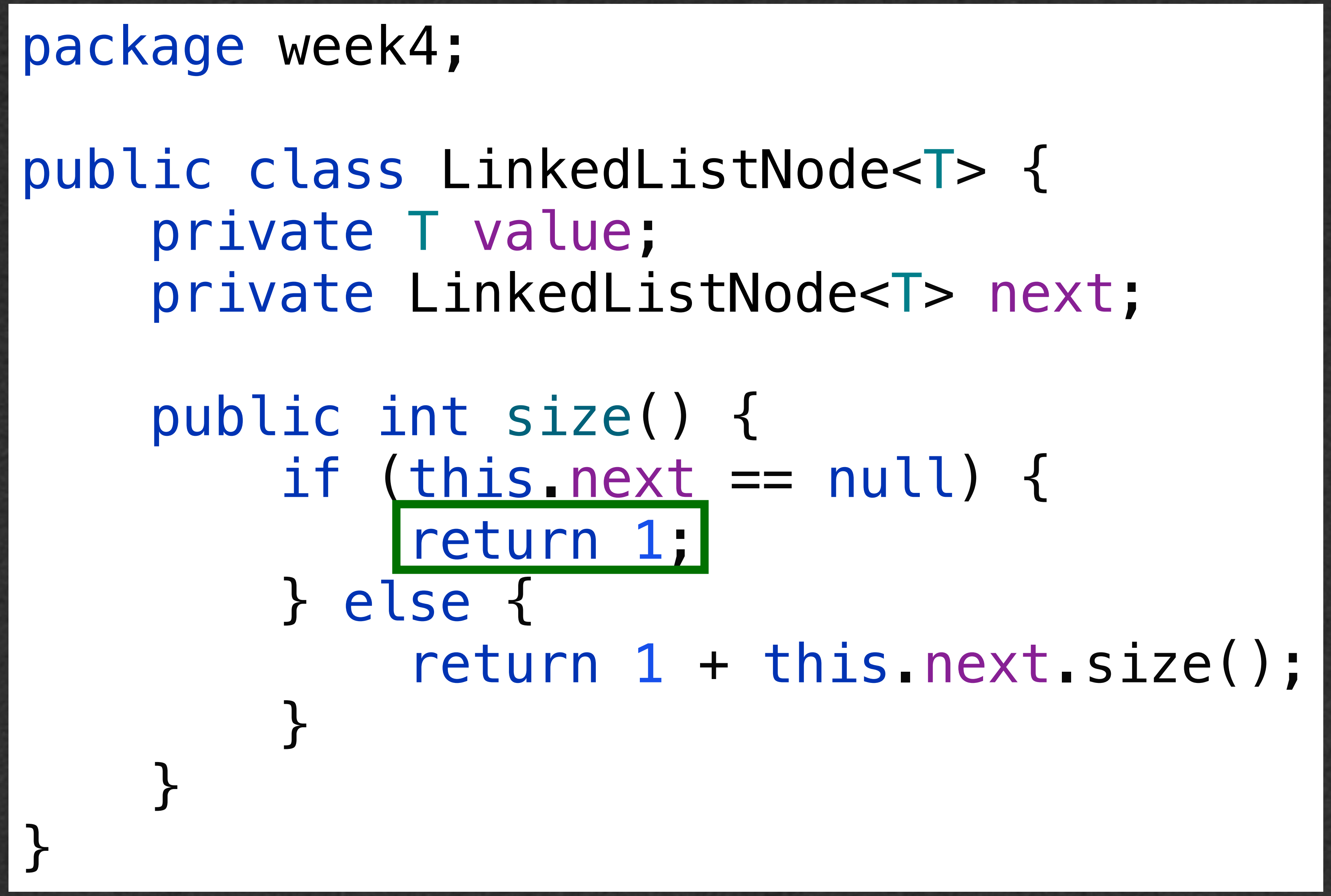
Node

1

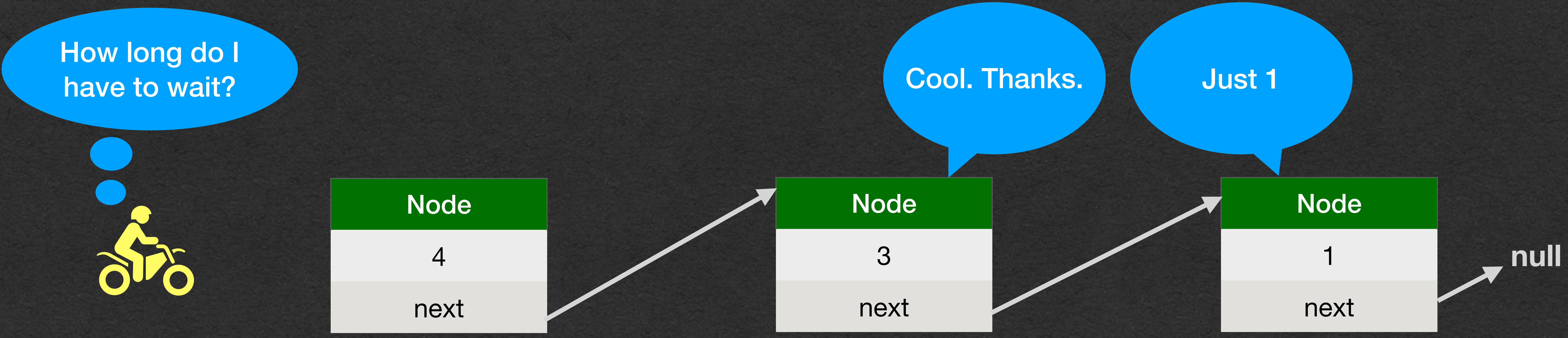
next

null

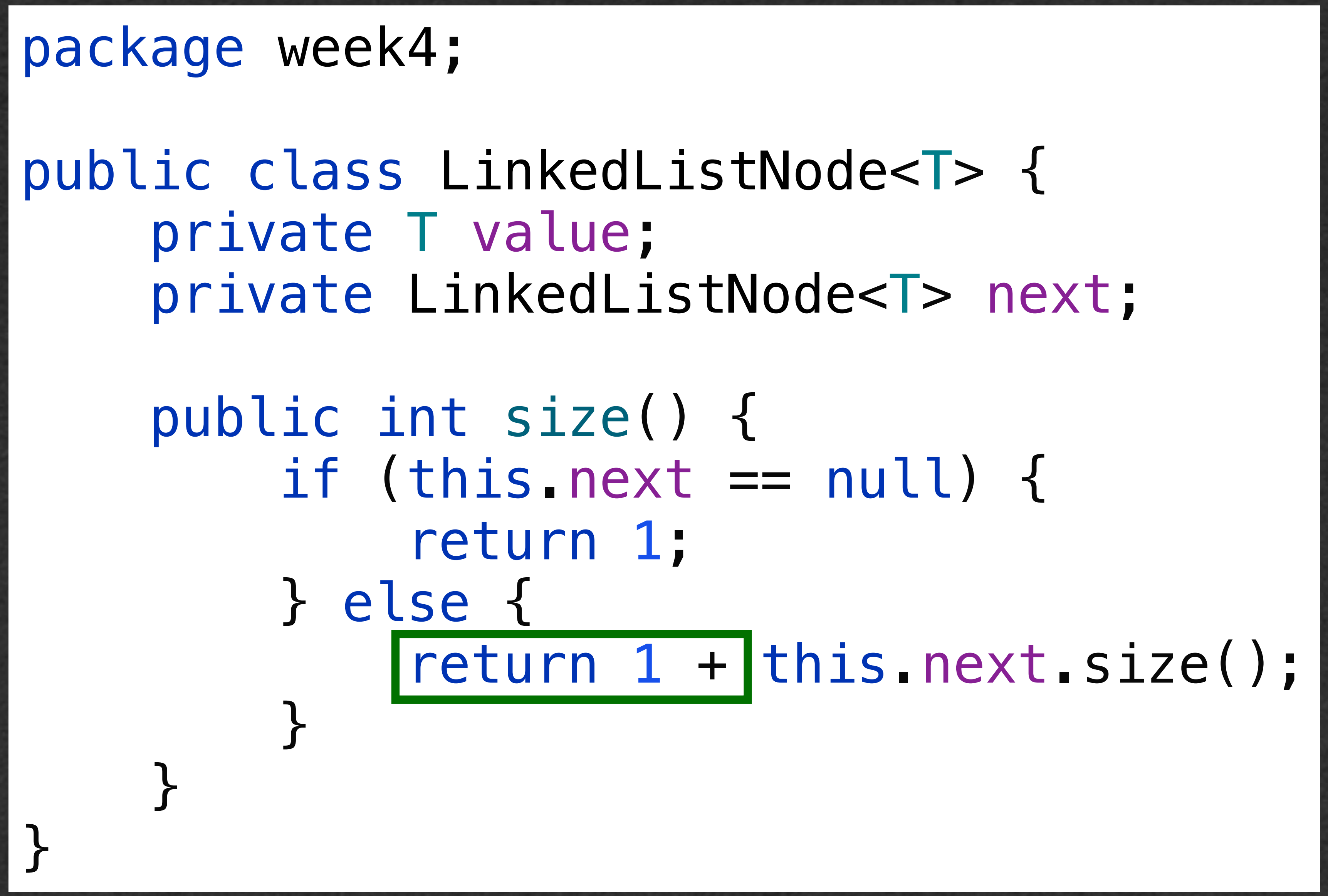
Linked List - Size



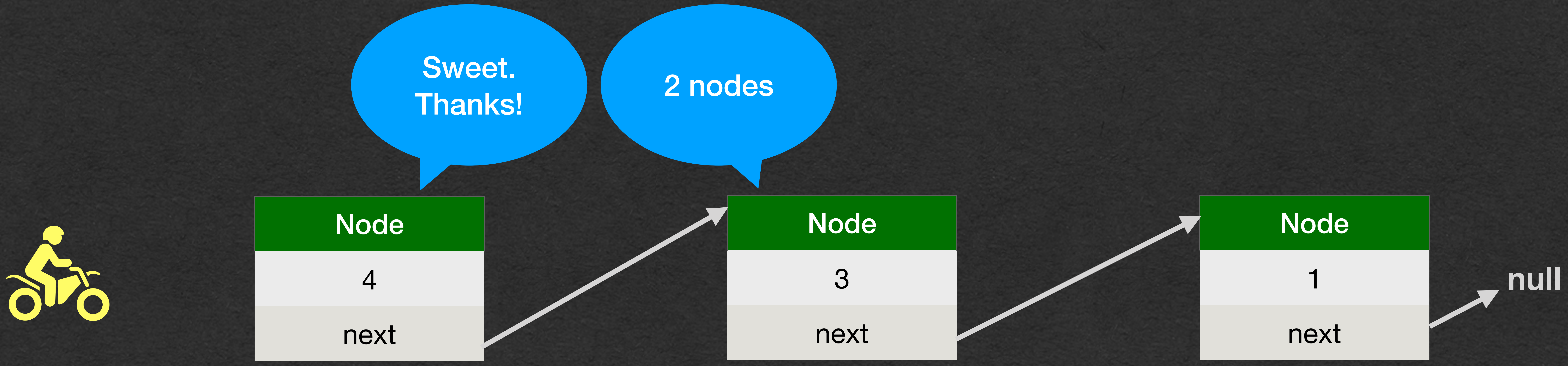
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



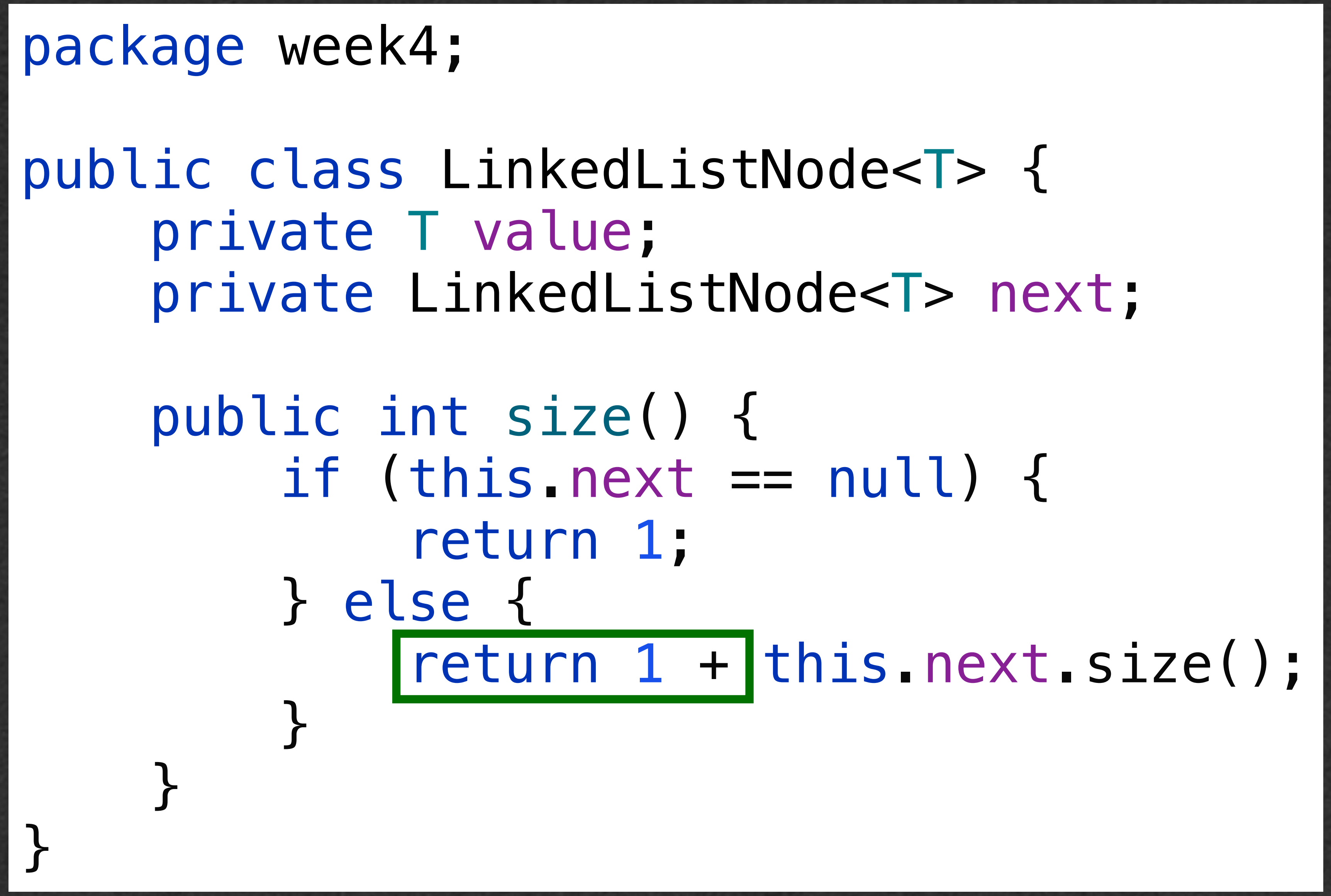
Linked List - Size



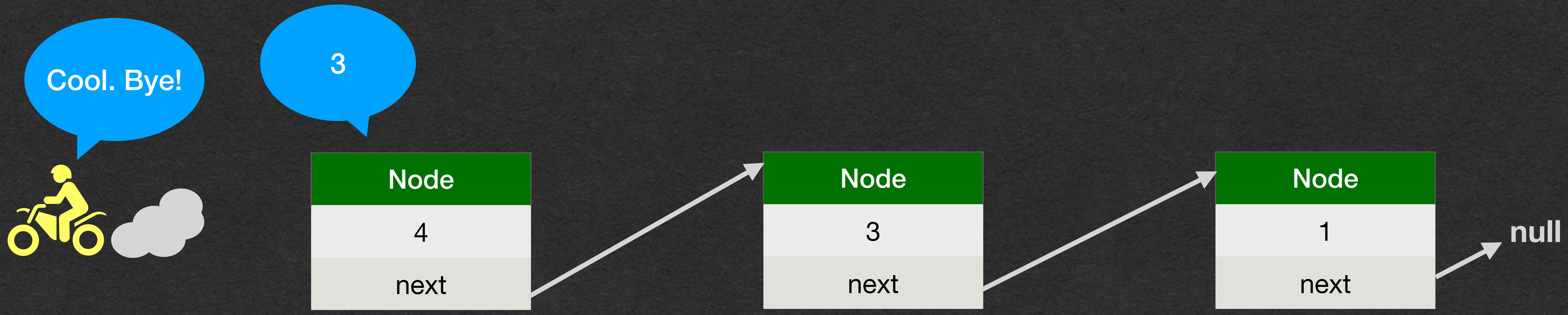
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



Linked List - Size

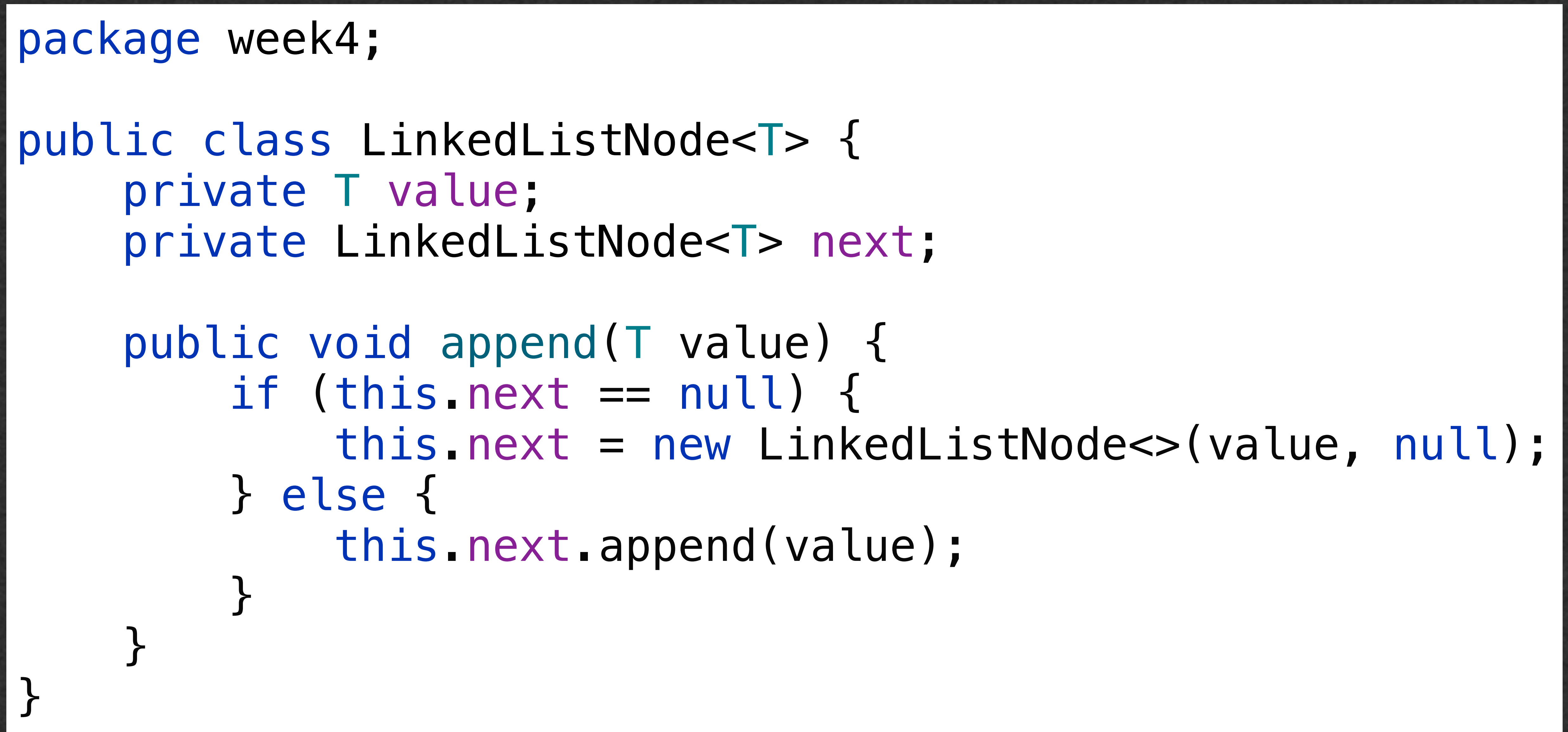


```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public int size() {
        if (this.next == null) {
            return 1;
        } else {
            return 1 + this.next.size();
        }
    }
}
```



Linked List - Append

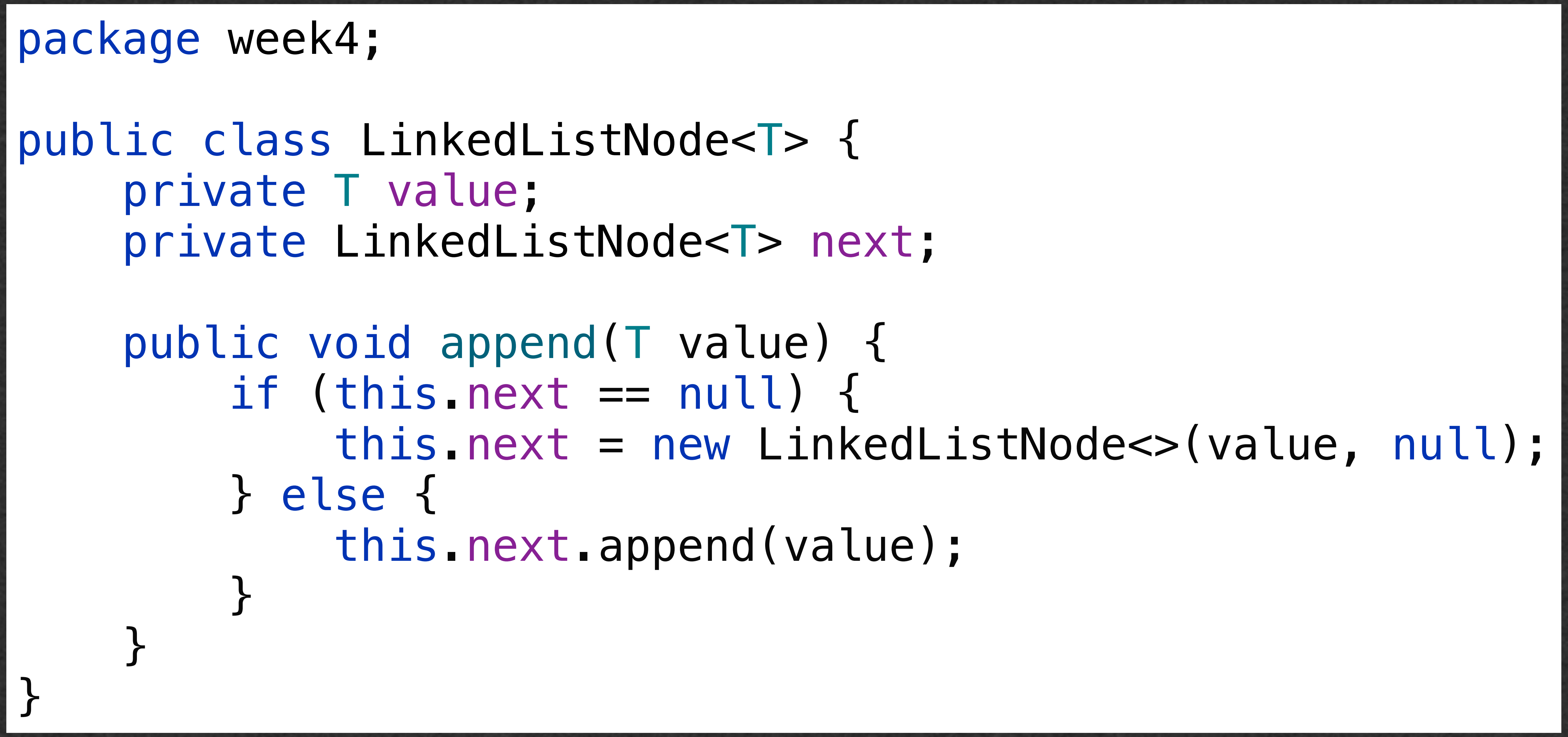
- Add an element to the end of the list
- First goal:
  - Find the end of the list
- When we find the last node:
  - Create a new node and set "next" of the last node to refer to the new node



```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```

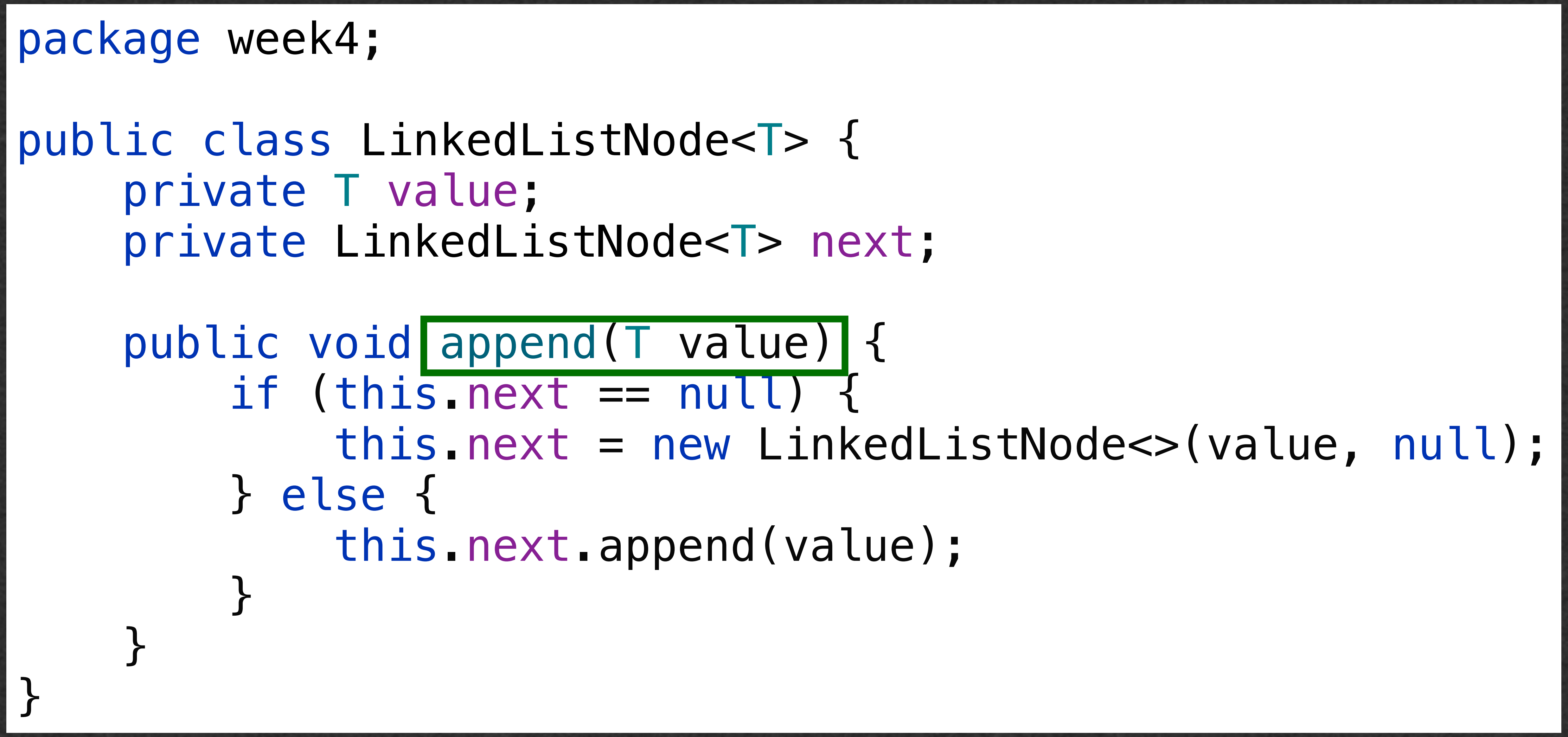
Linked List - Append

- If next is null
  - We're at the last node
  - Add the new node here
- If next is not null
  - Make a recursive call on the next node to move down the list

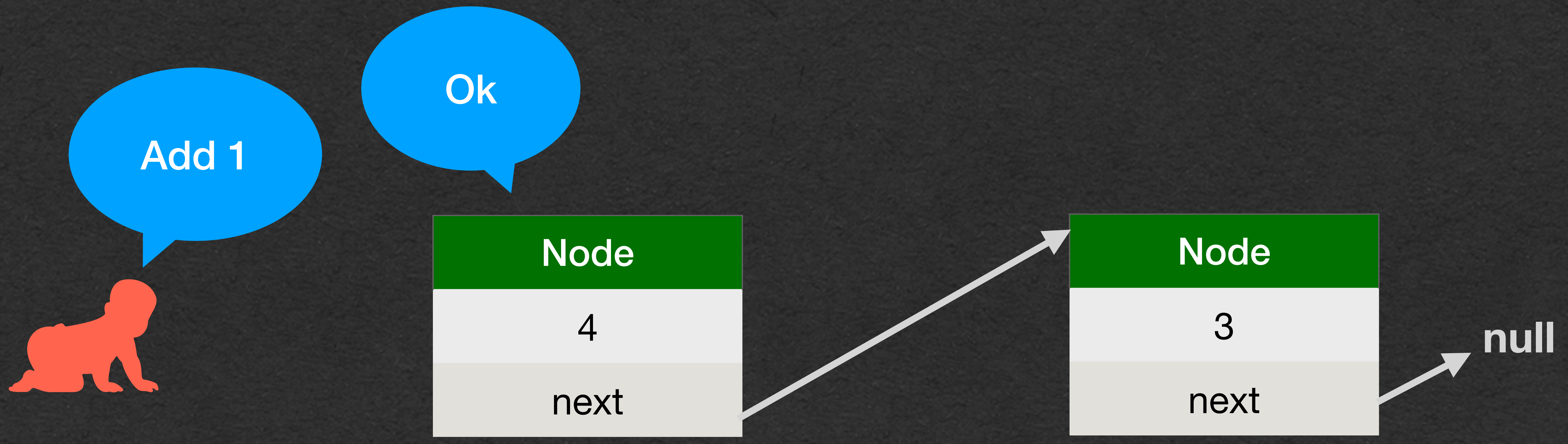


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```

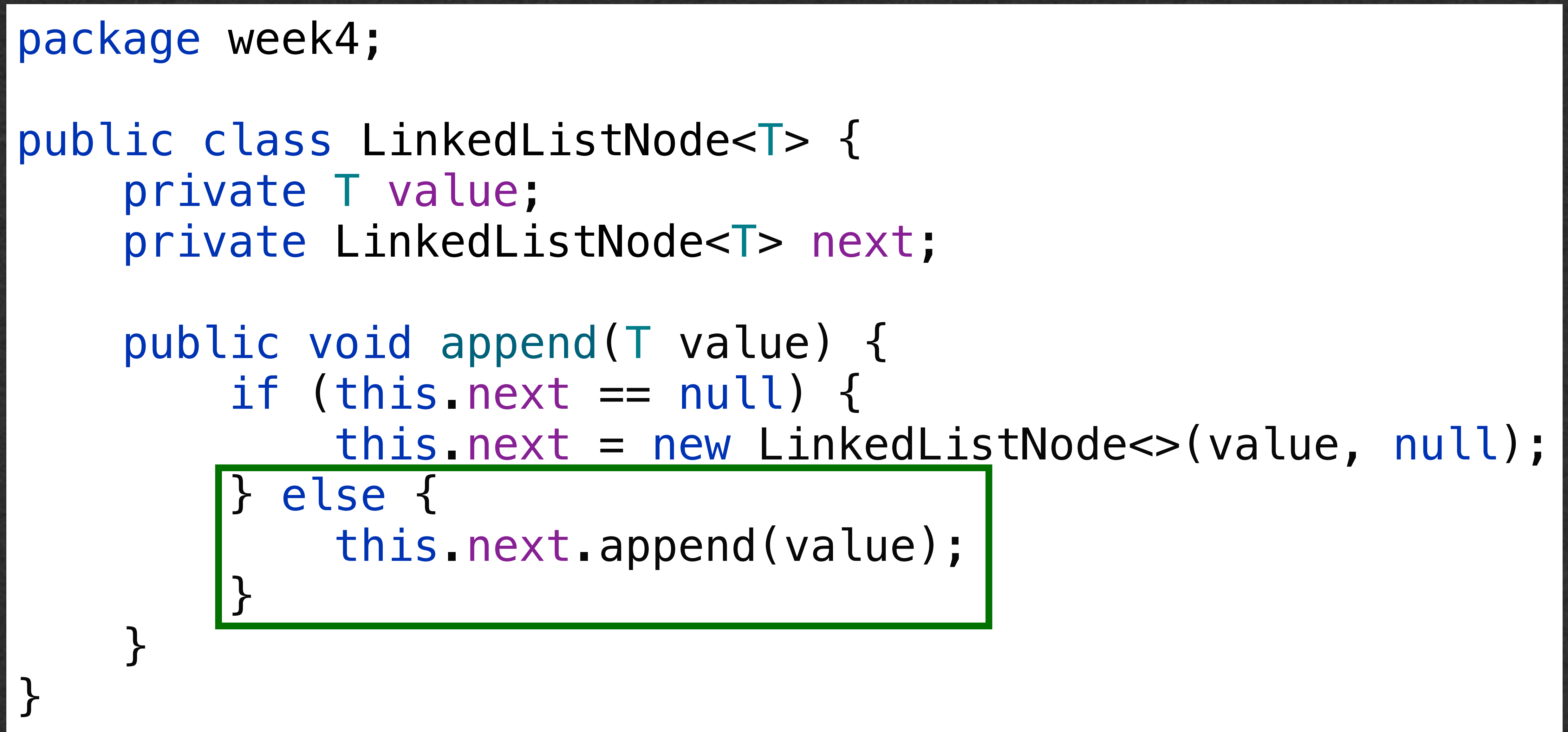
Linked List - Append



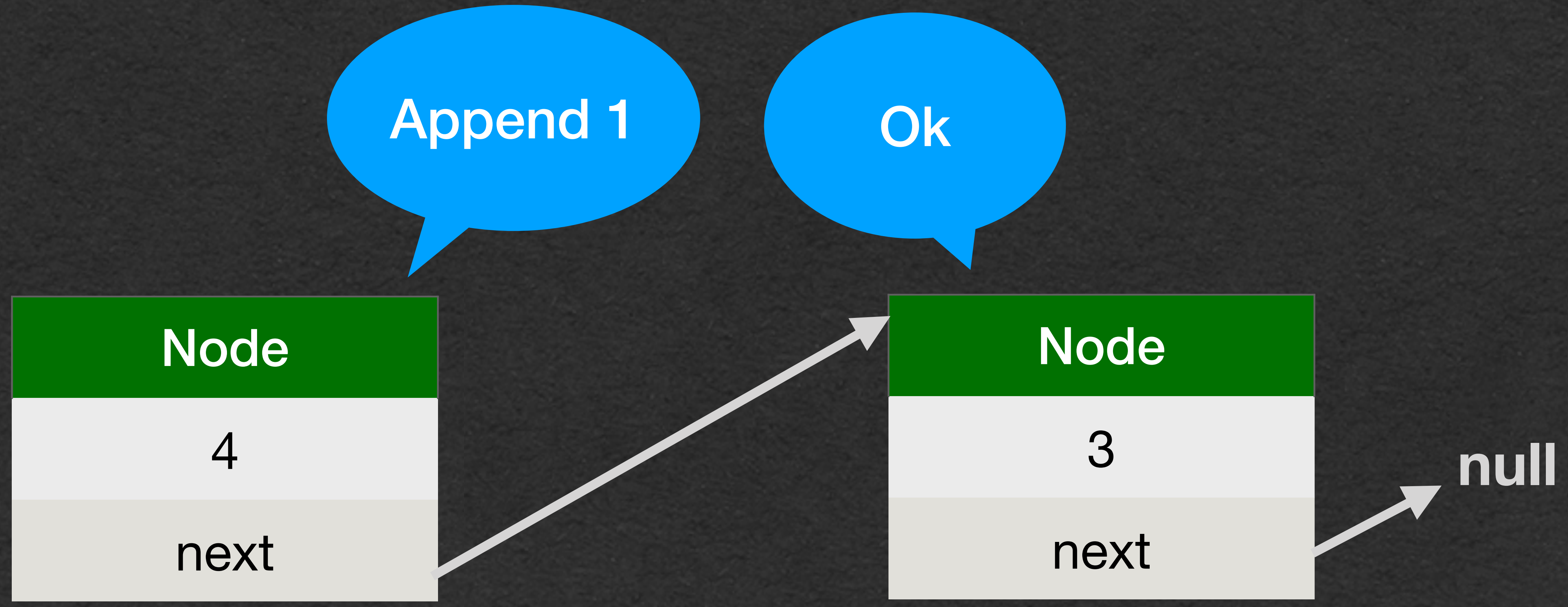
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```



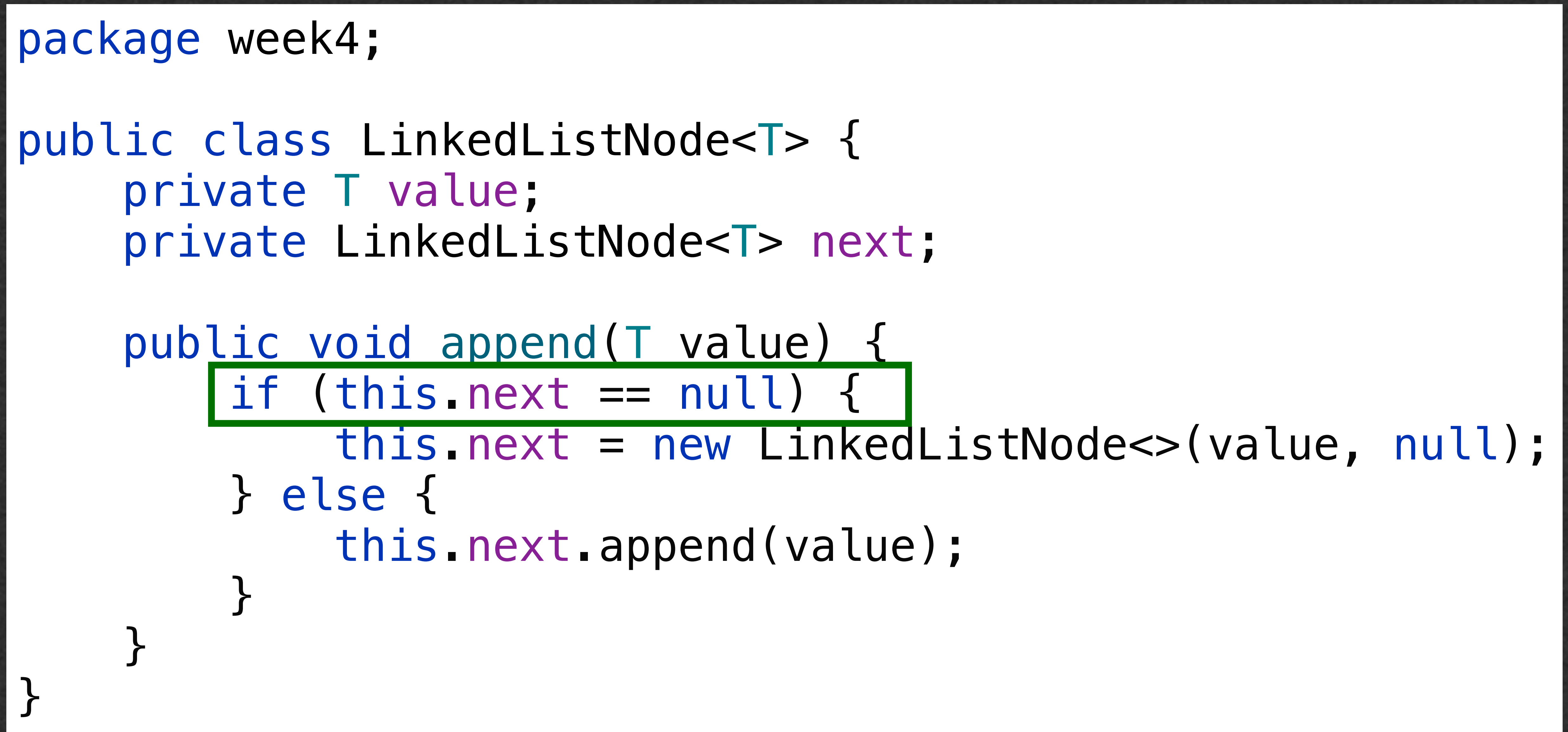
Linked List - Append



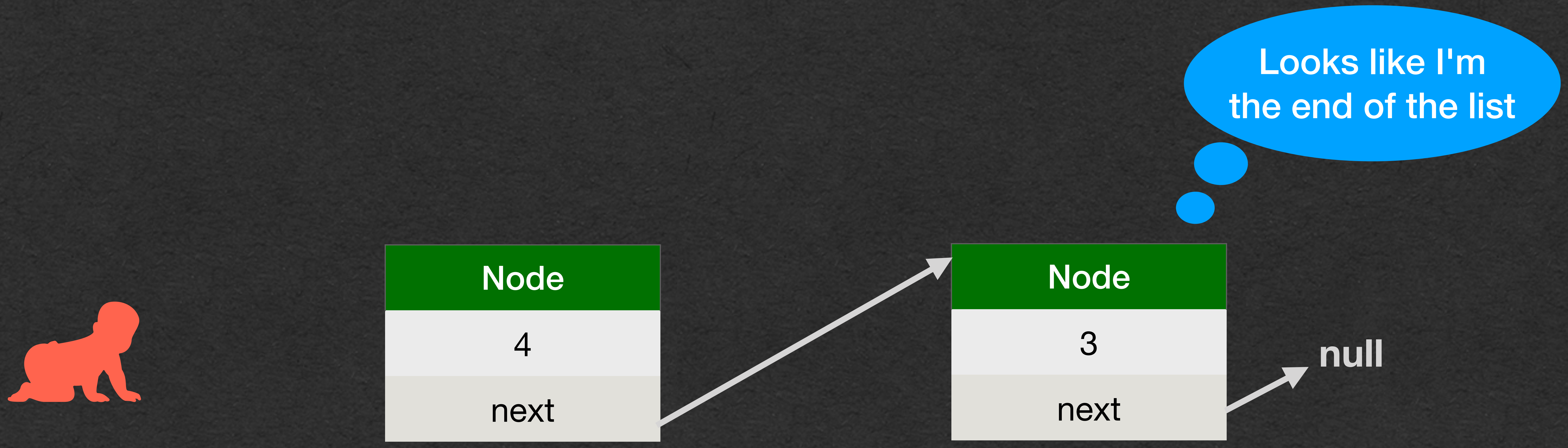
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```



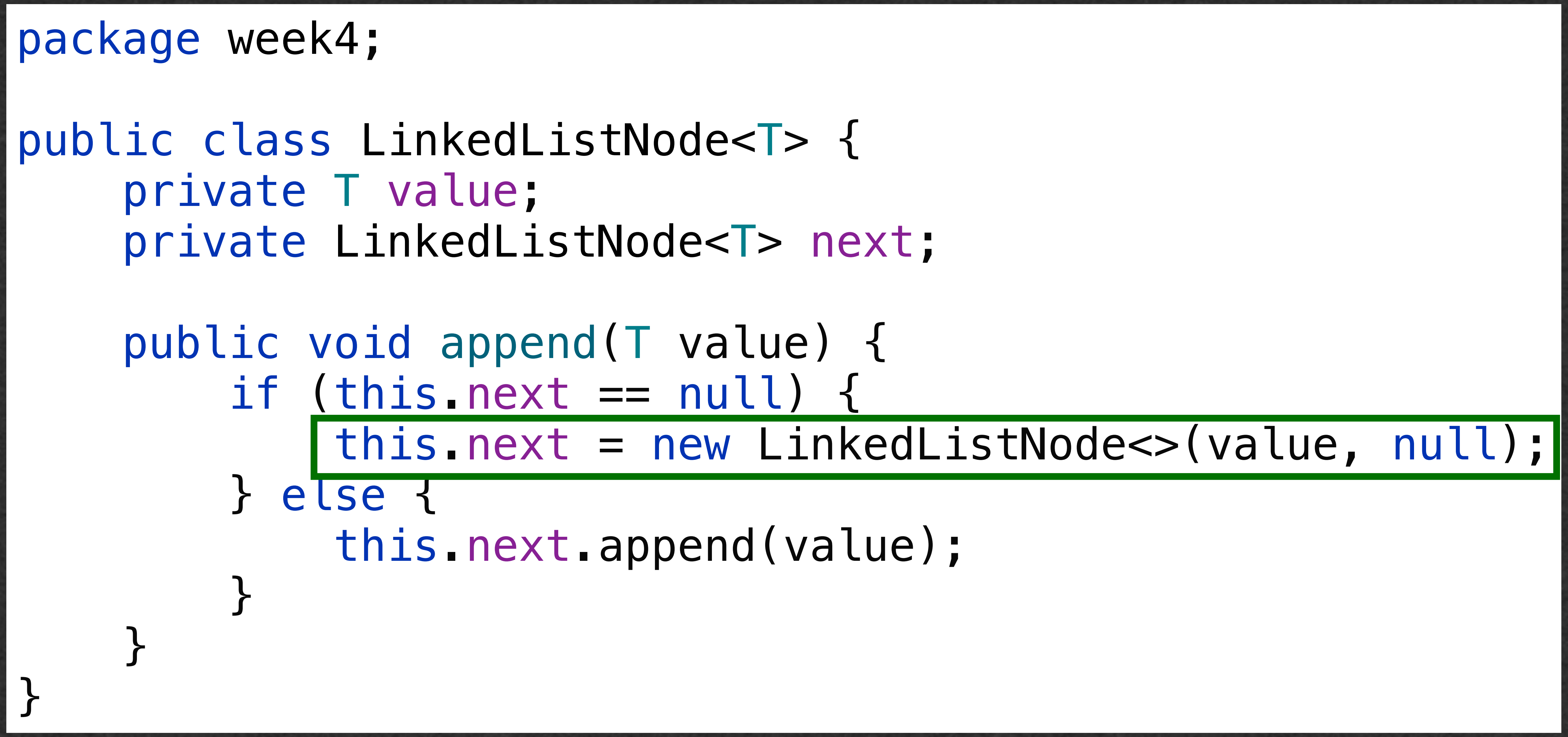
Linked List - Append



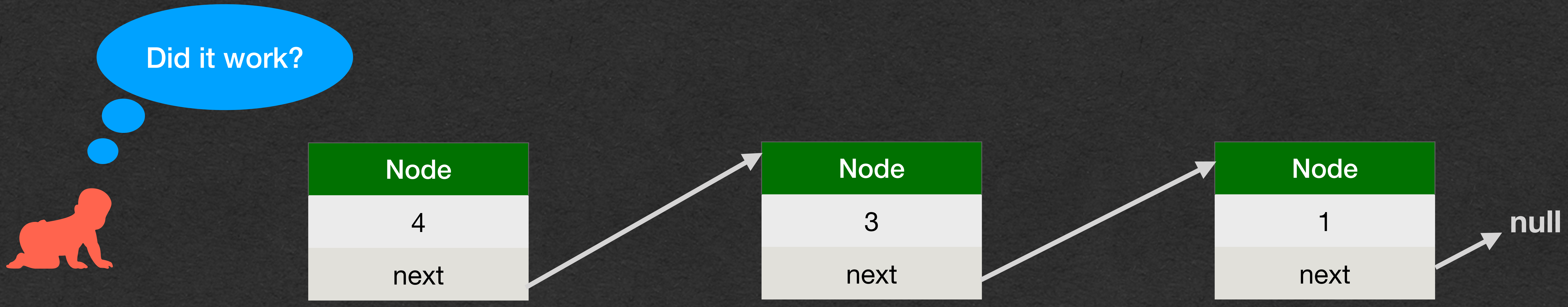
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```



Linked List - Append



```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public void append(T value) {
        if (this.next == null) {
            this.next = new LinkedListNode<>(value, null);
        } else {
            this.next.append(value);
        }
    }
}
```



Did it work?

Node

4

next

Node

3

next

Node

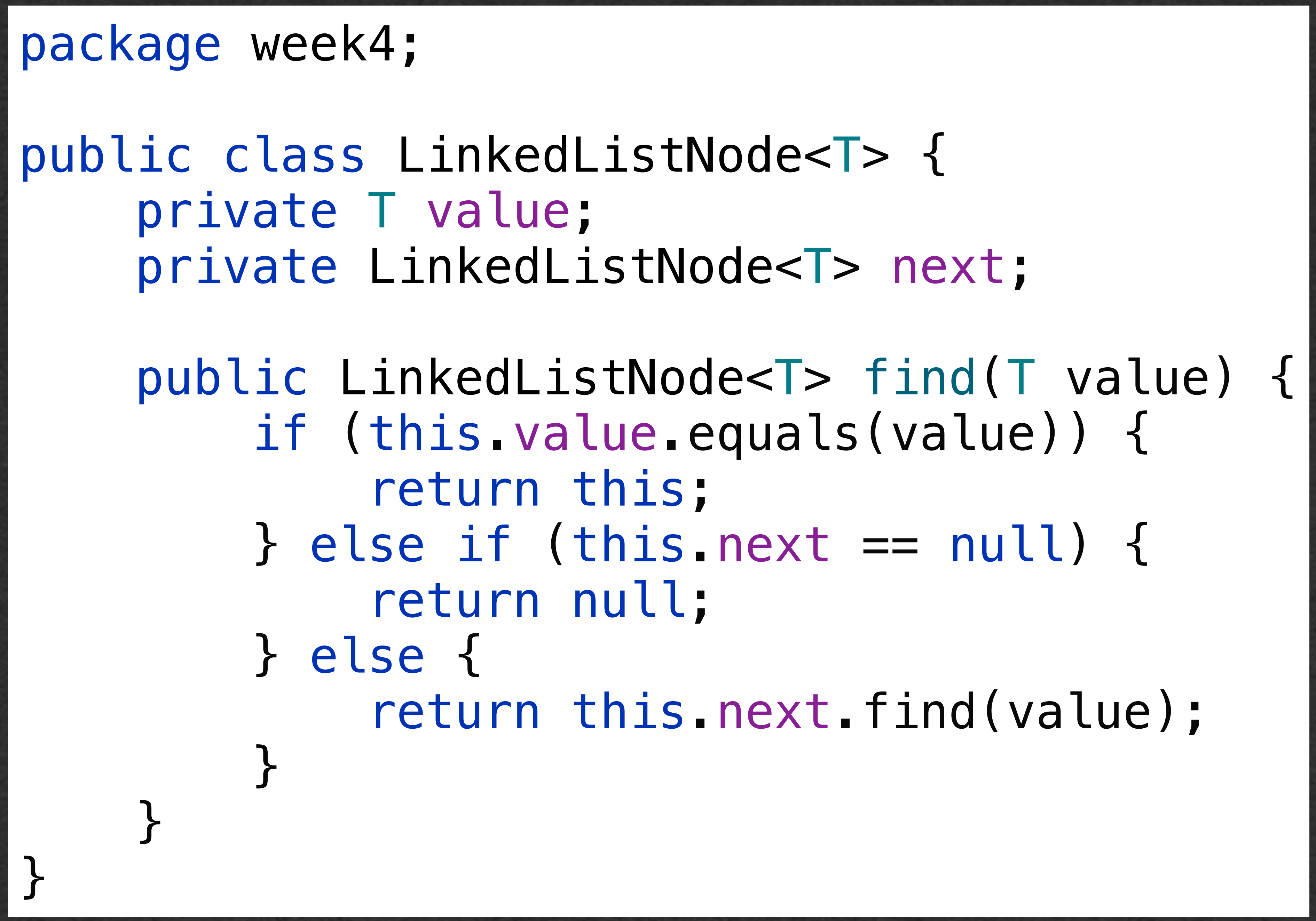
1

next

null

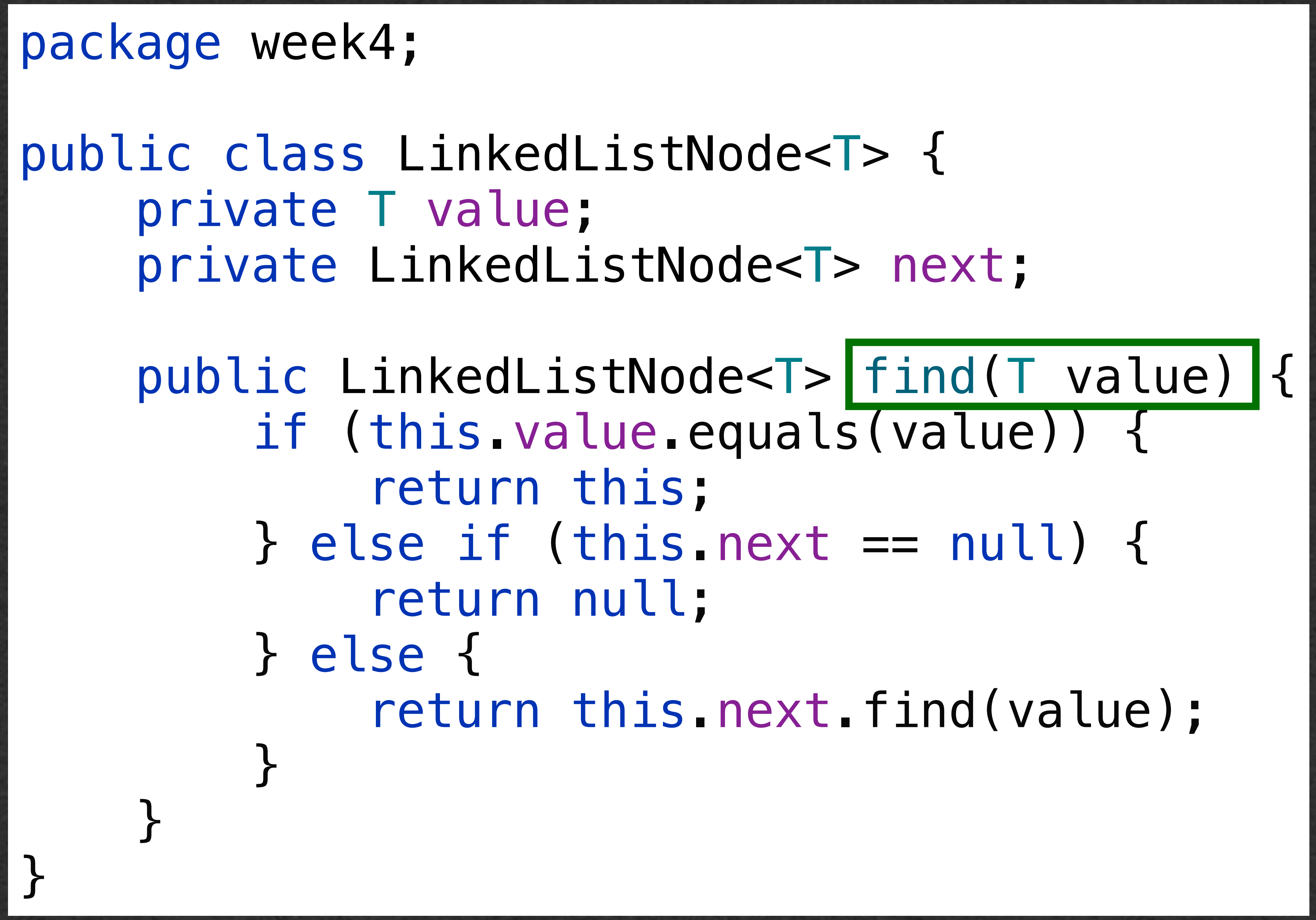
Linked List - Find

- Navigate through the list one node at a time
  - Check if the node contains the value
  - If it doesn't, move to the next node
  - If the end of the list is reached, the list does not contain the element

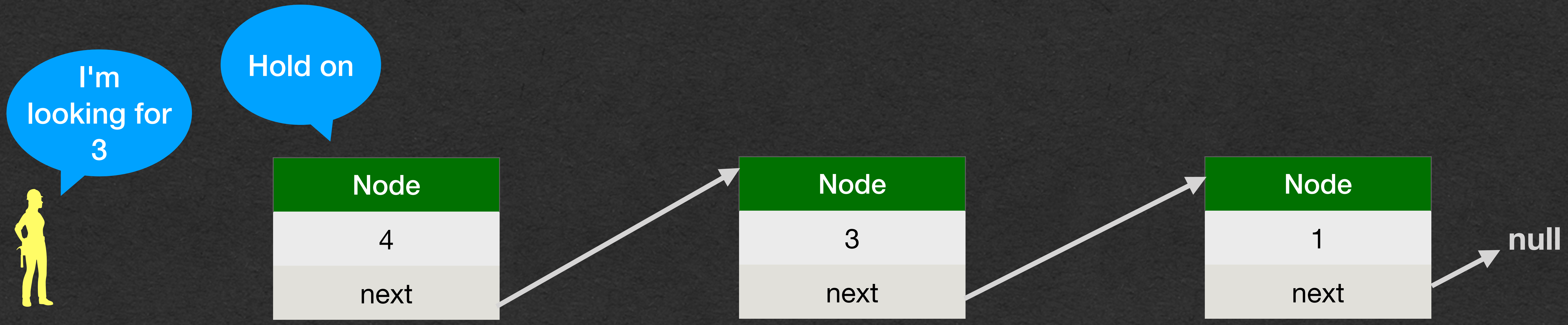


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```

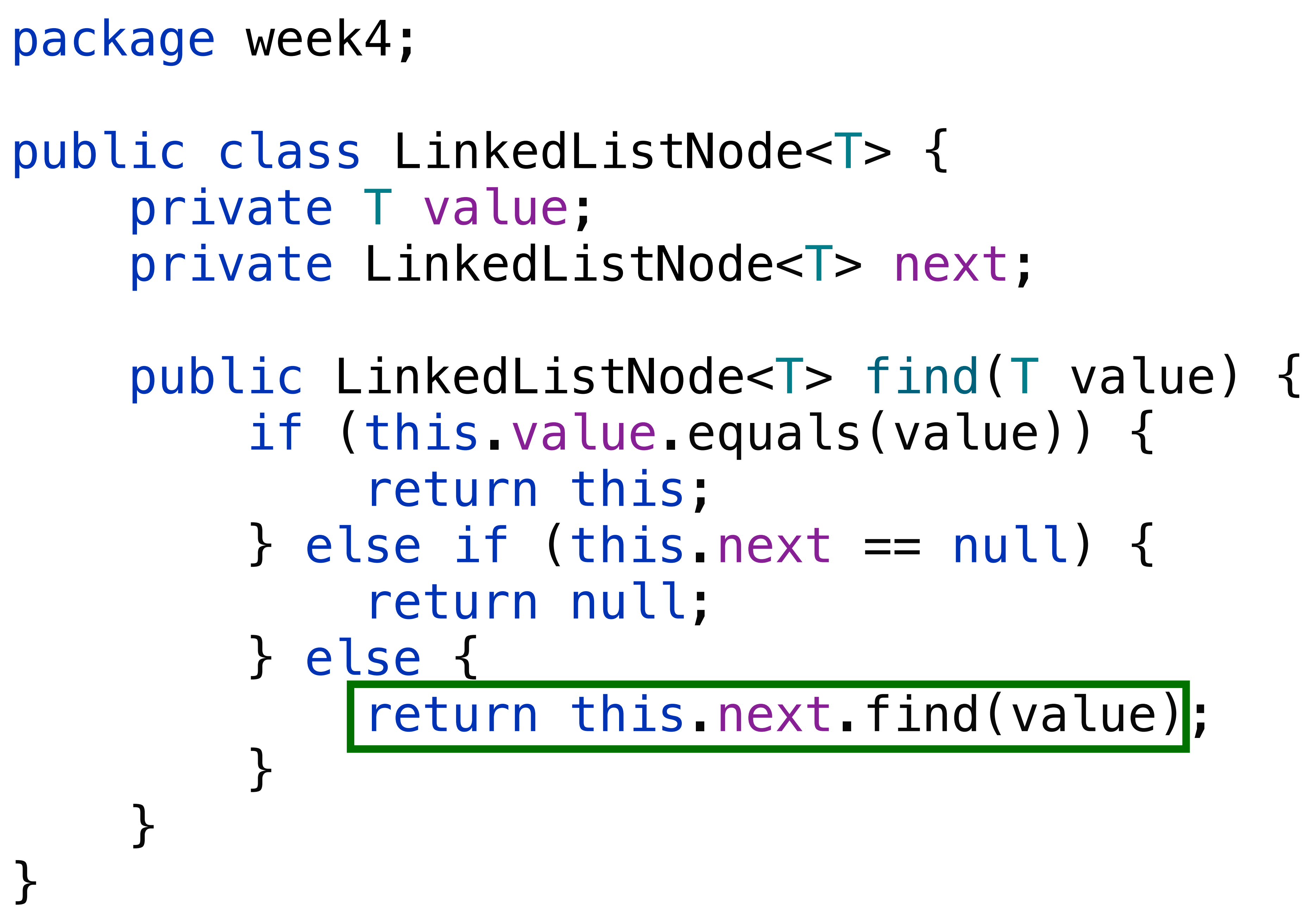
Linked List - Find



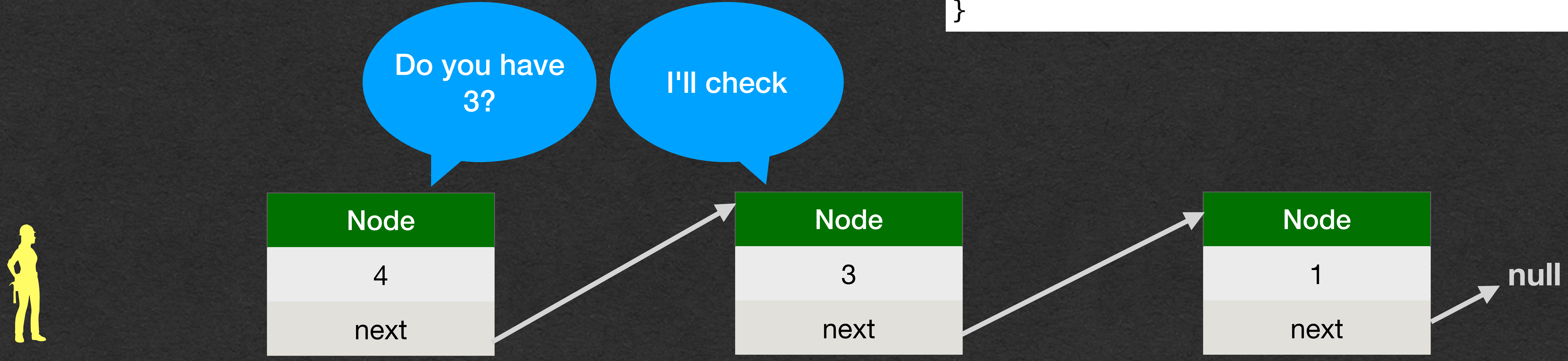
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



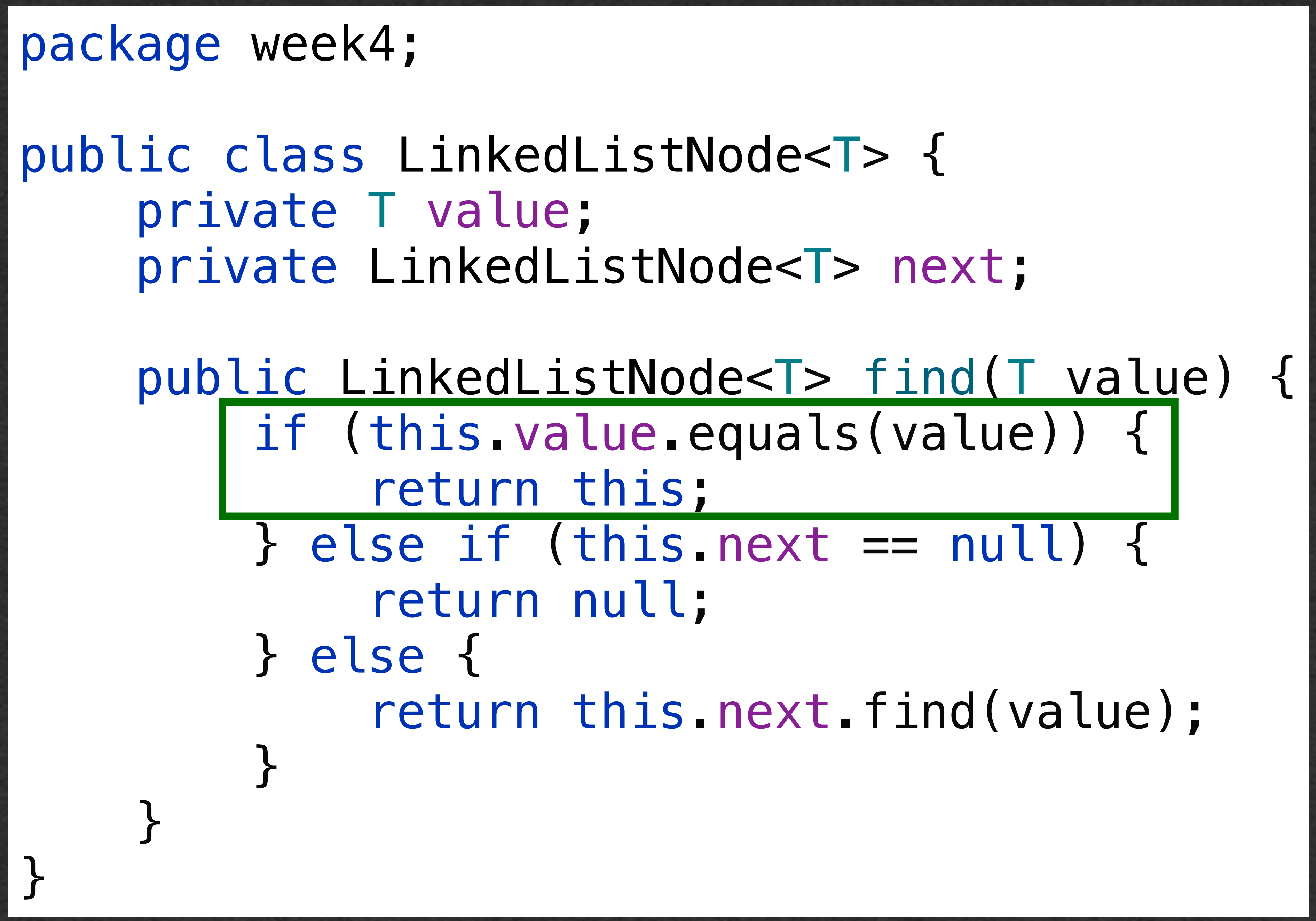
Linked List - Find



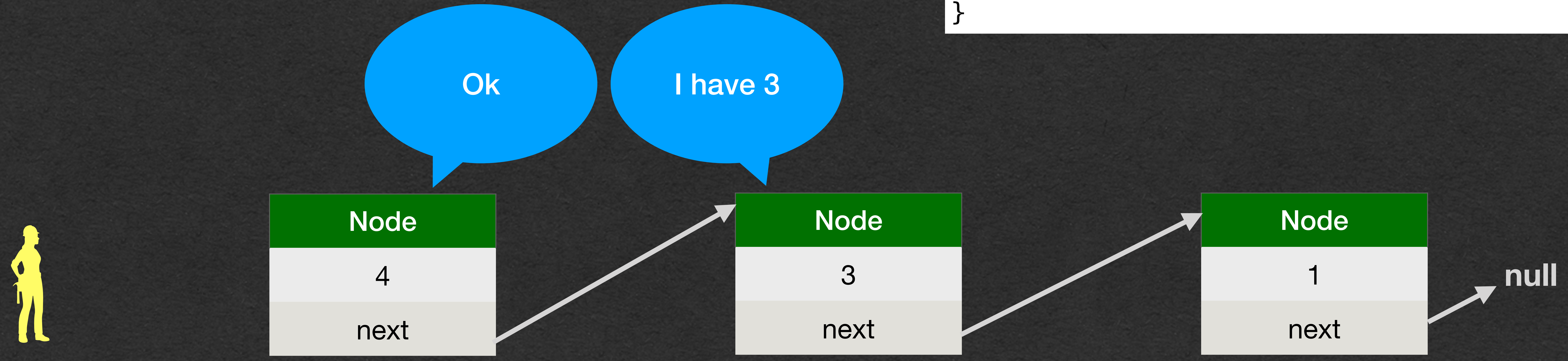
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



Linked List - Find

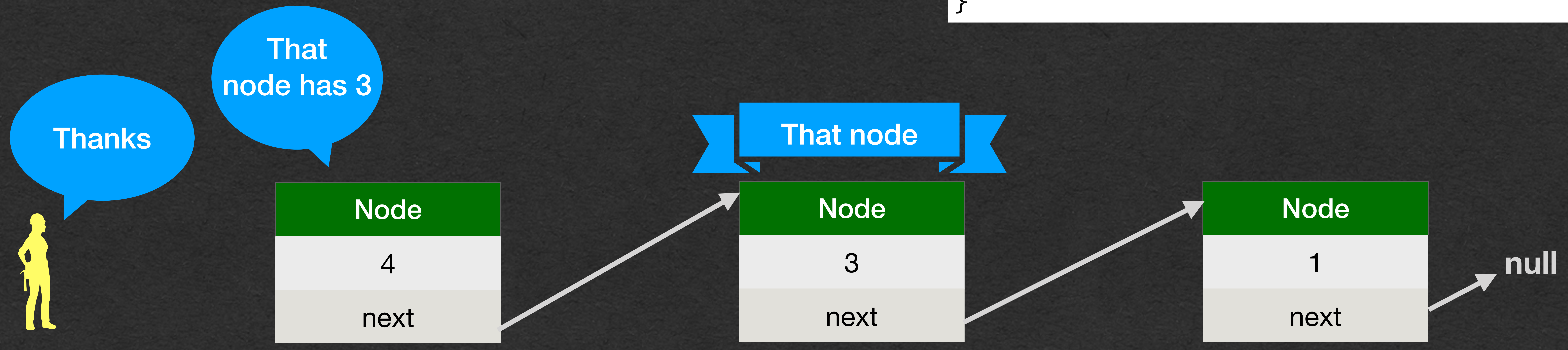


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```

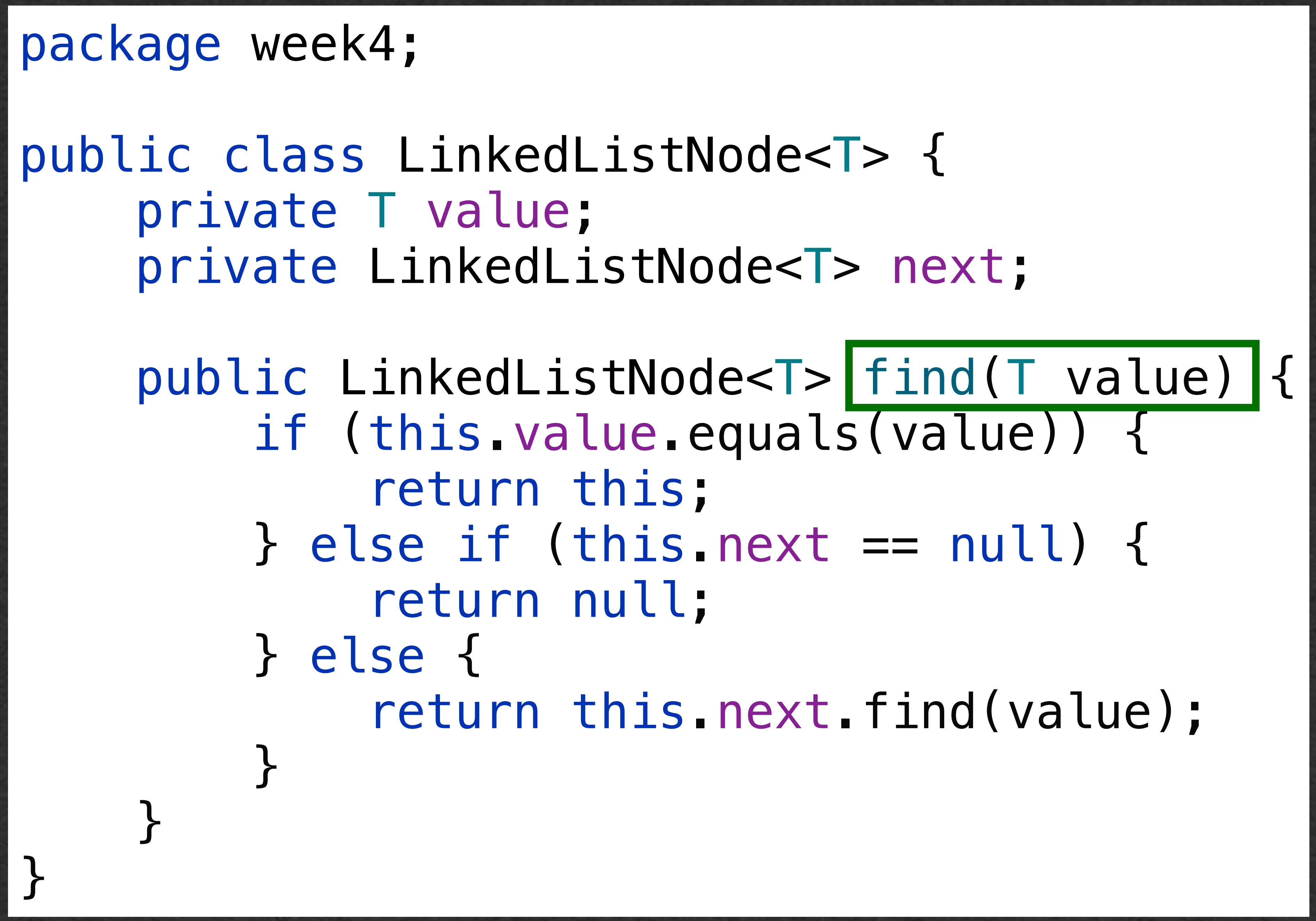


Linked List - Find

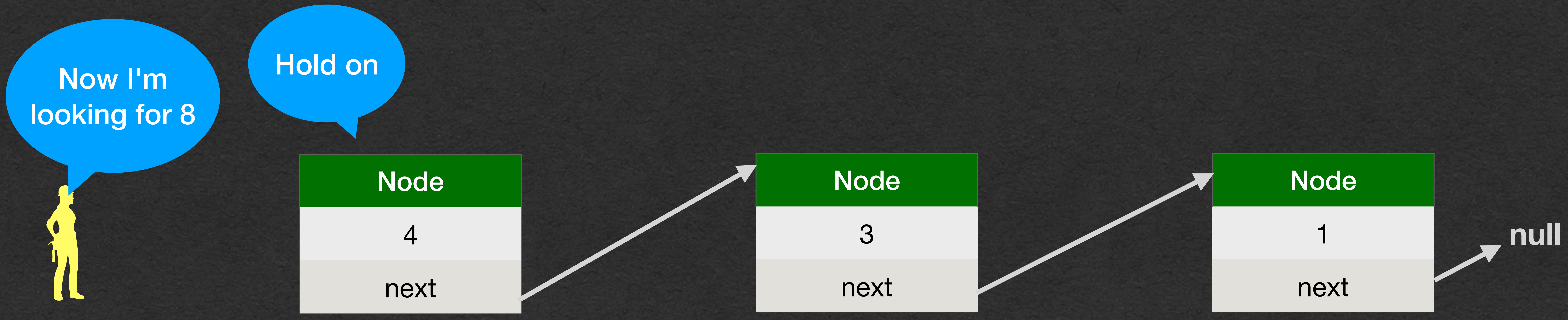
```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



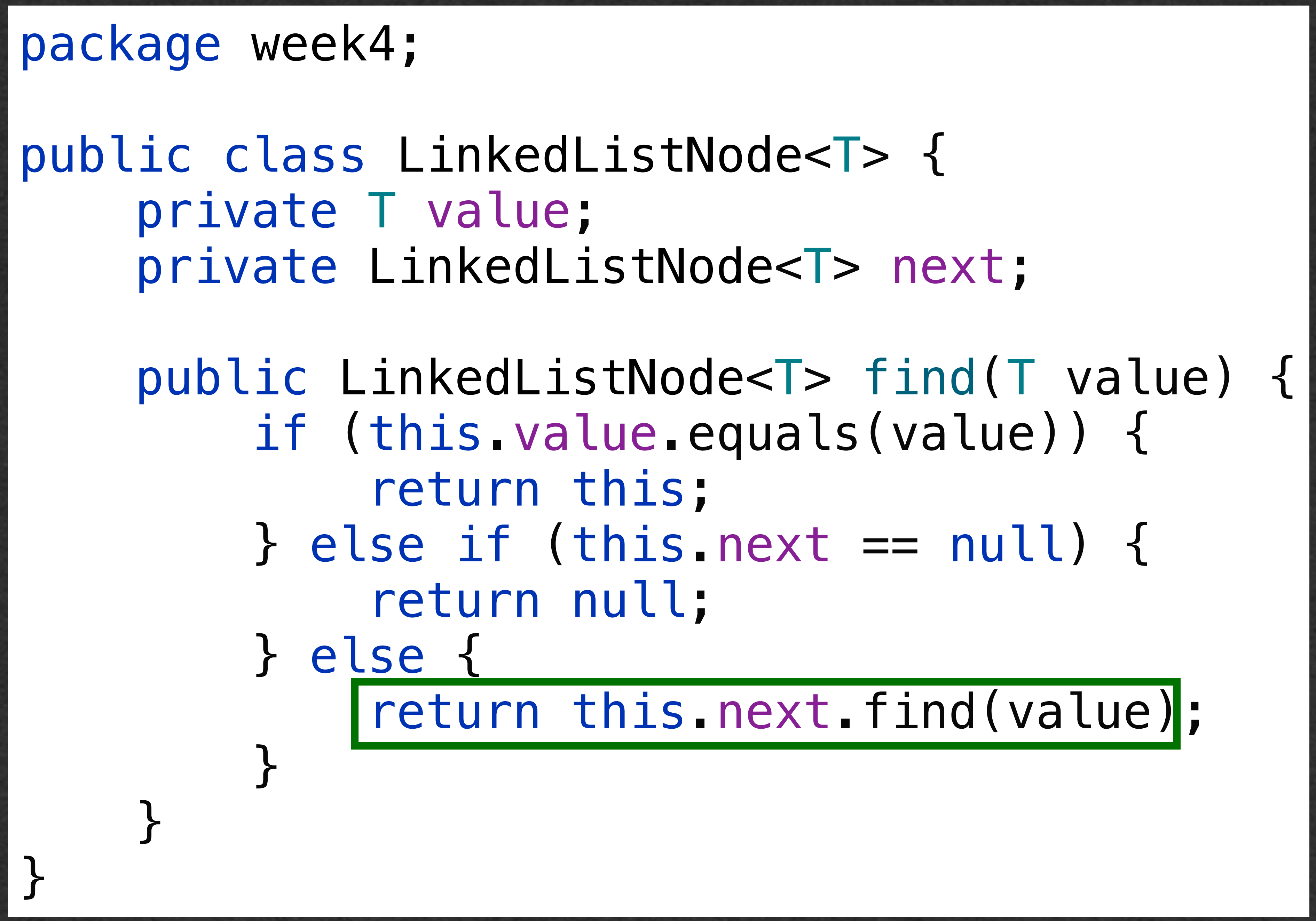
Linked List - Find



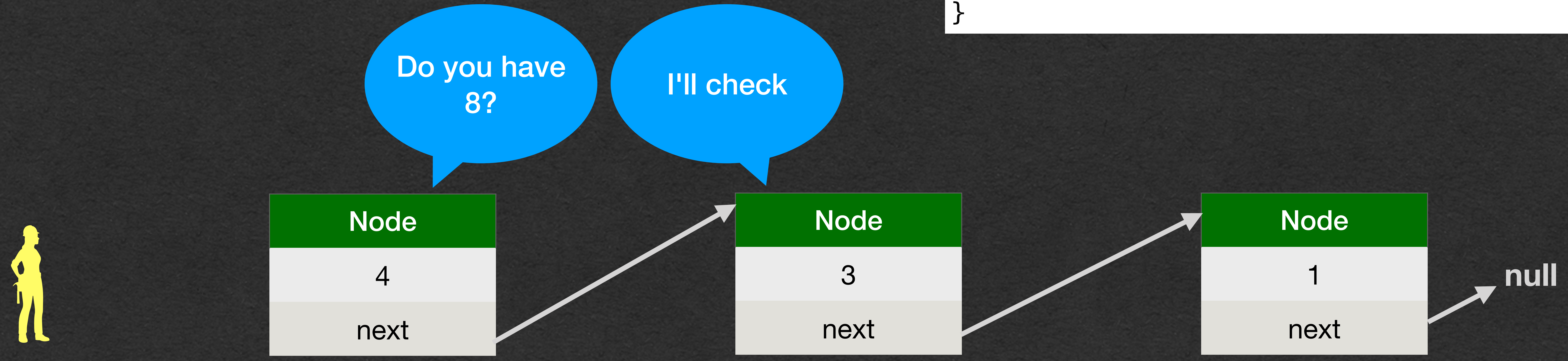
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



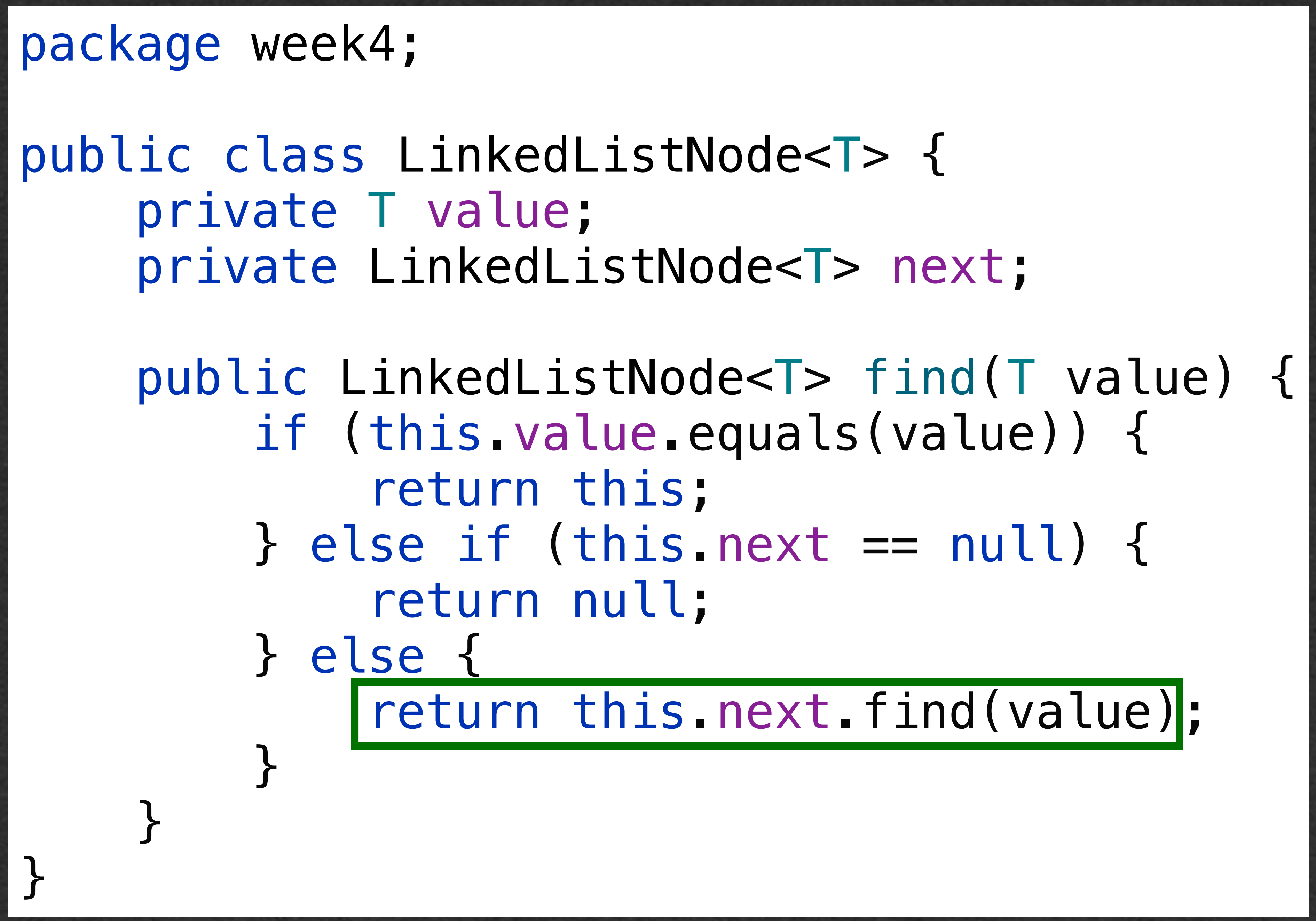
Linked List - Find



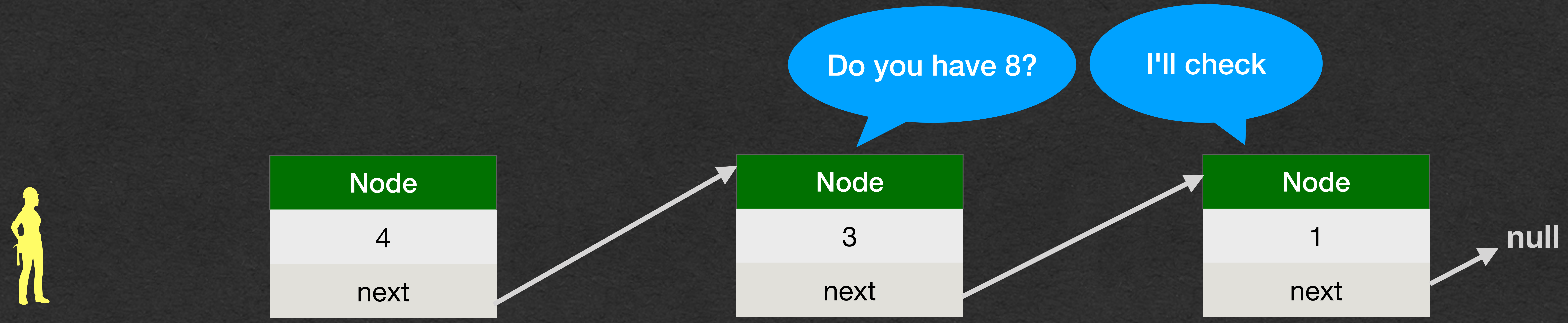
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



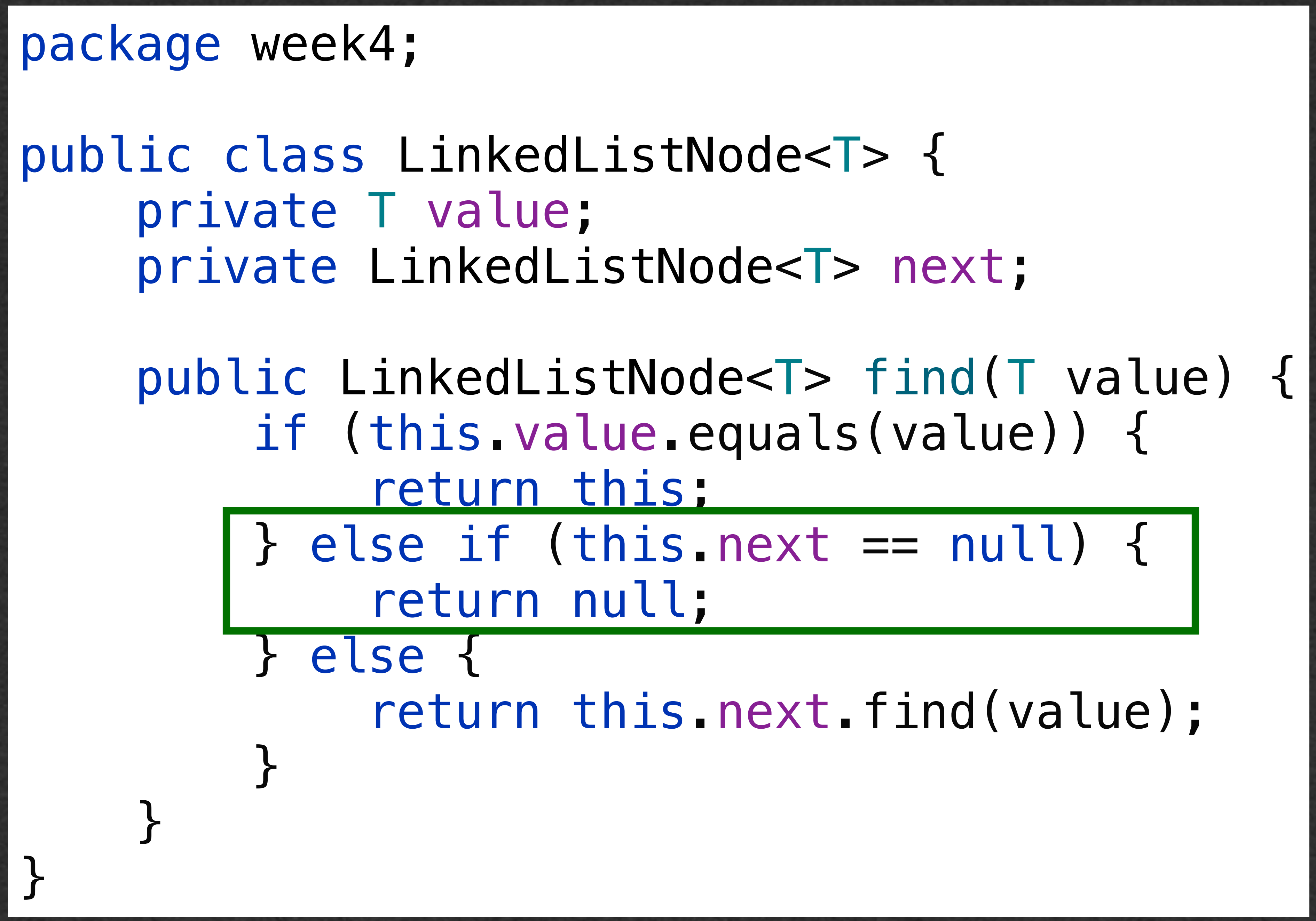
Linked List - Find



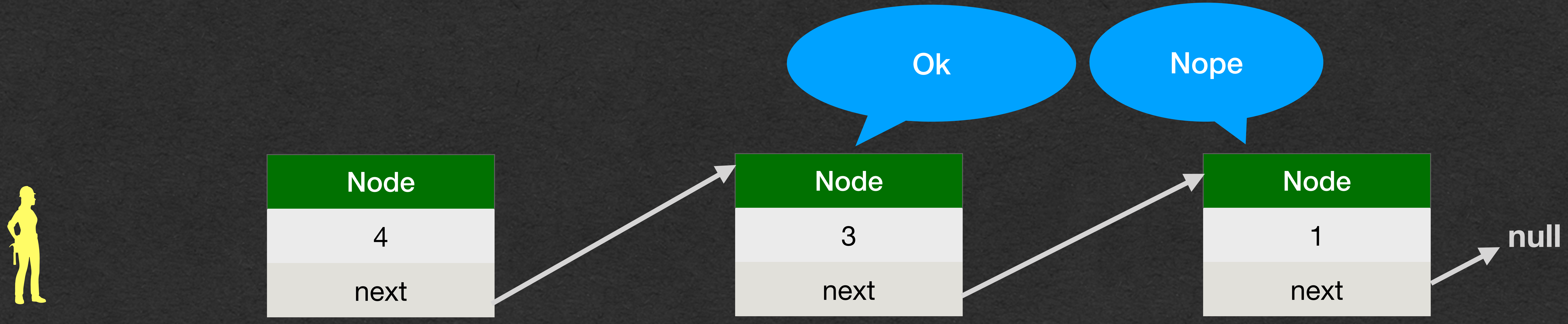
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



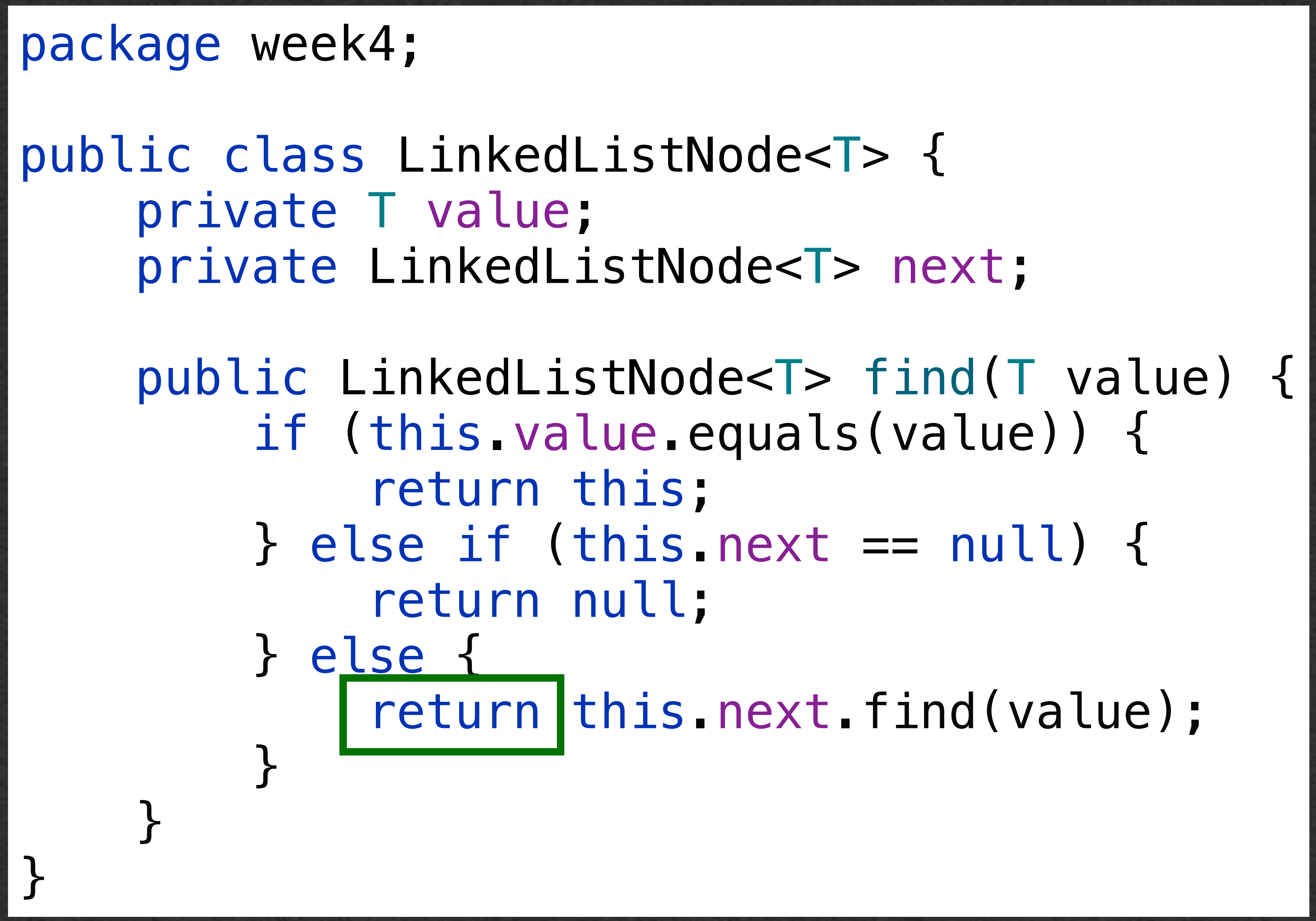
Linked List - Find



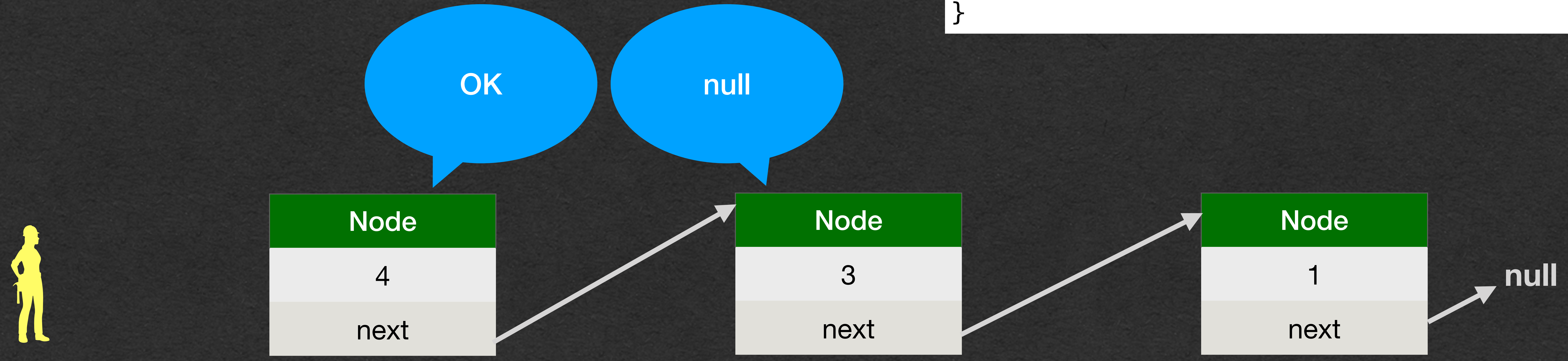
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



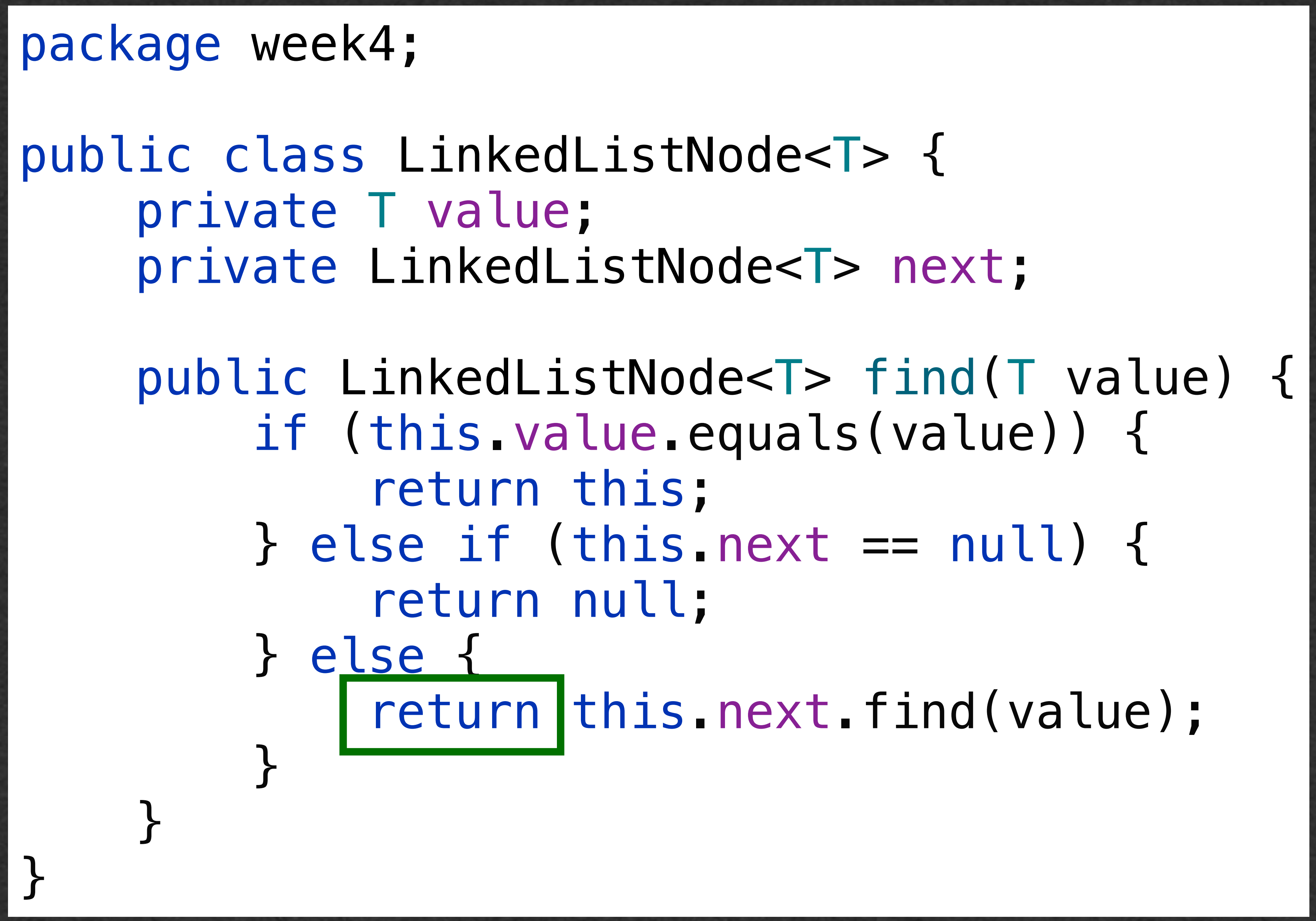
Linked List - Find



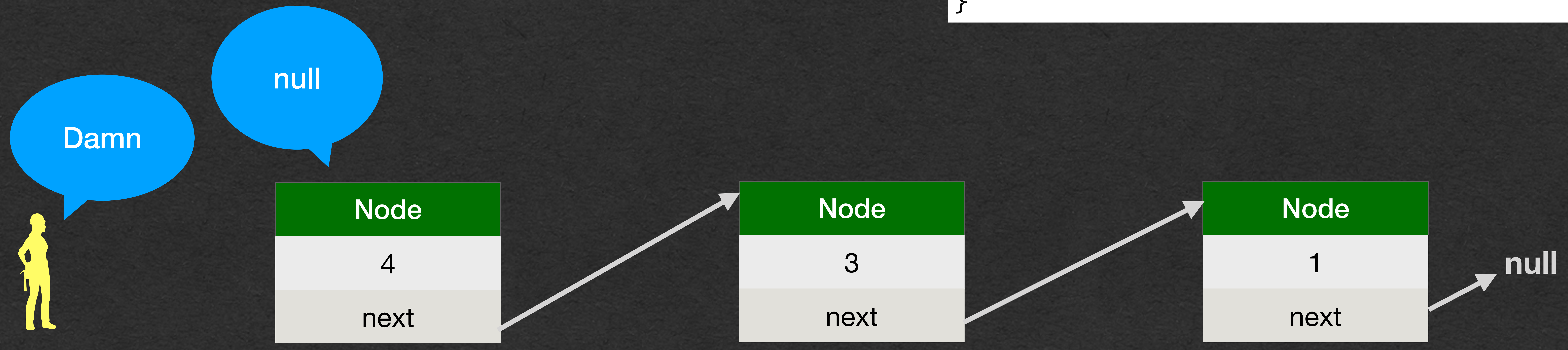
```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



Linked List - Find

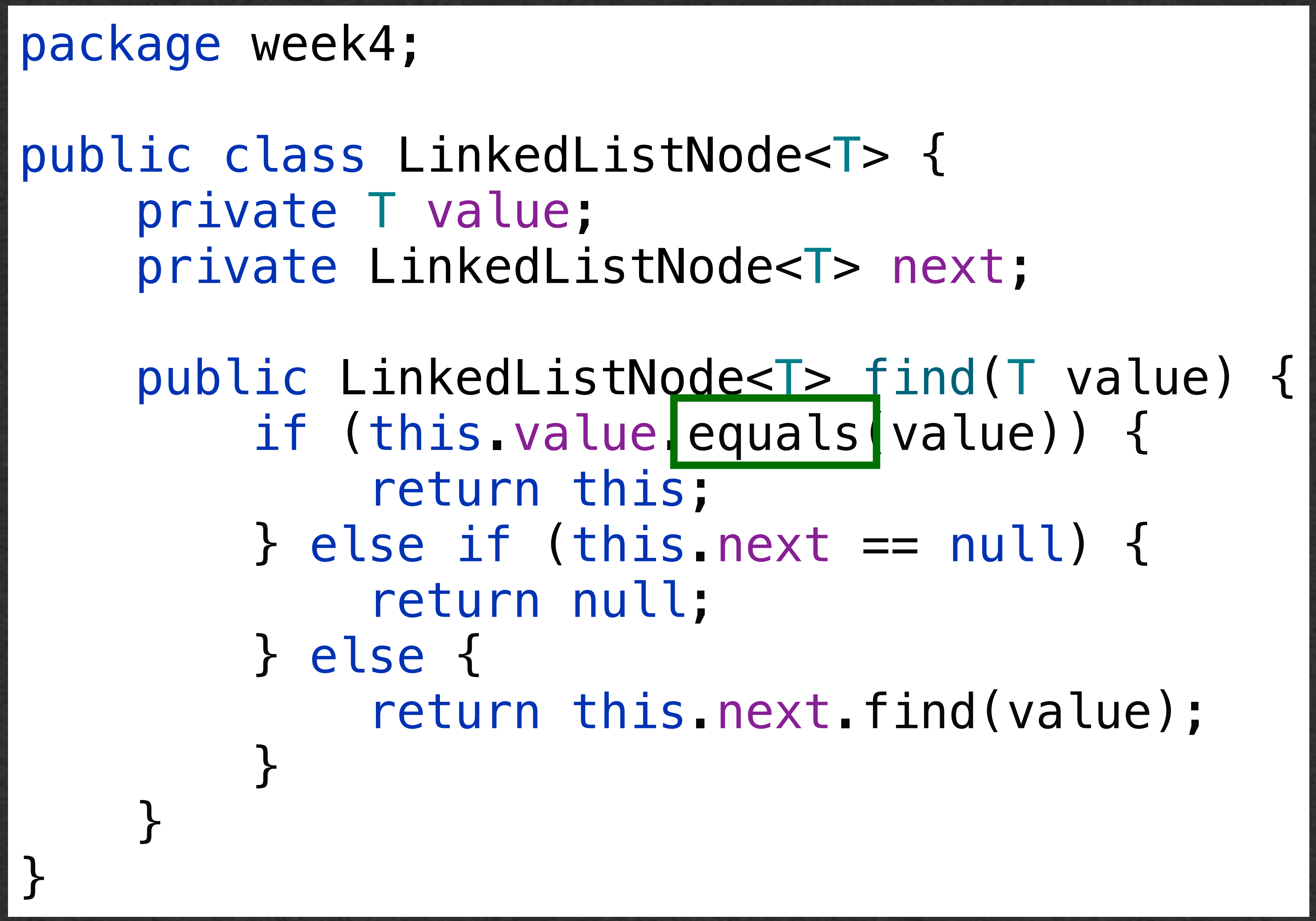


```
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```



Linked List - Min?

- Find worked for us even though we're using generics
  - Every class has an equals method!
  - Can call equals on any type (Except primitives)
  - With generics we can only call method that every class has (toString, equals, hashCode)
  - We can't do much else with these values

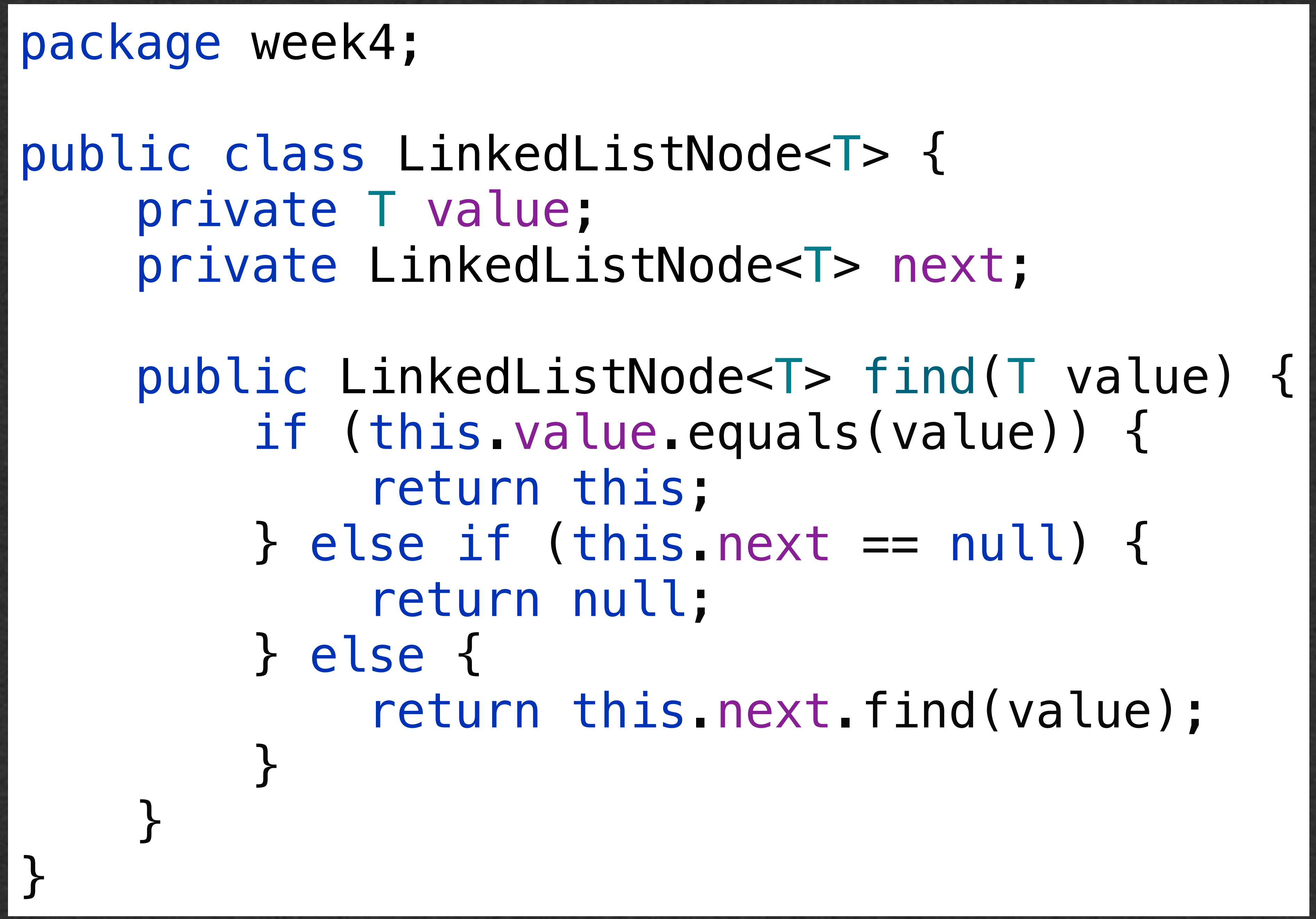


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```

Linked List - Min?

- What if we want to find the min value in a list of Doubles?
  - Not all classes have a less than operator

- Or what if we want to find a Rating made by a specific Reviewer?..

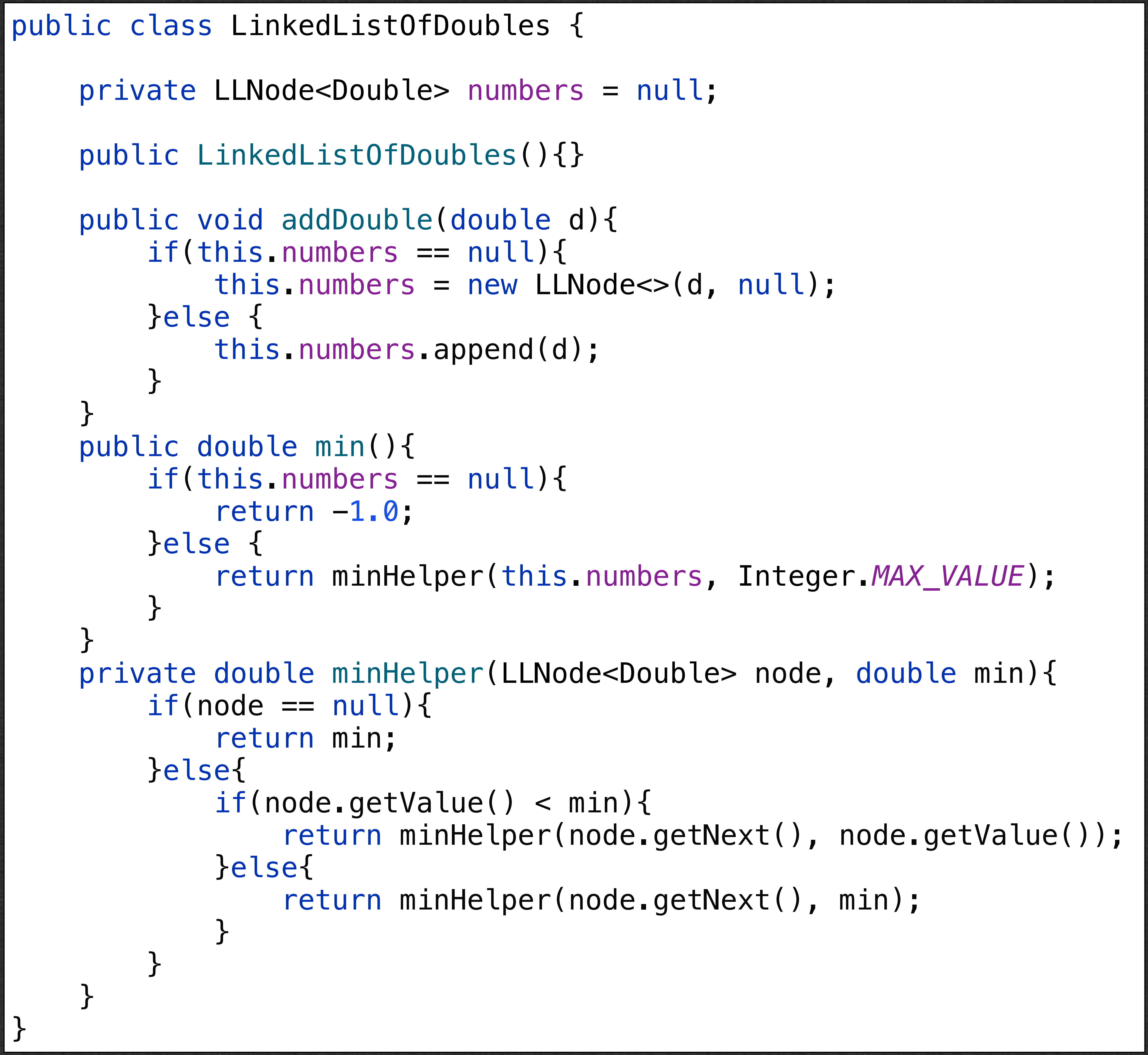


```java
package week4;

public class LinkedListNode<T> {
    private T value;
    private LinkedListNode<T> next;

    public LinkedListNode<T> find(T value) {
        if (this.value.equals(value)) {
            return this;
        } else if (this.next == null) {
            return null;
        } else {
            return this.next.find(value);
        }
    }
}
```

Linked List - Min

- Let's look at an example that contains a Linked List in an instance variable

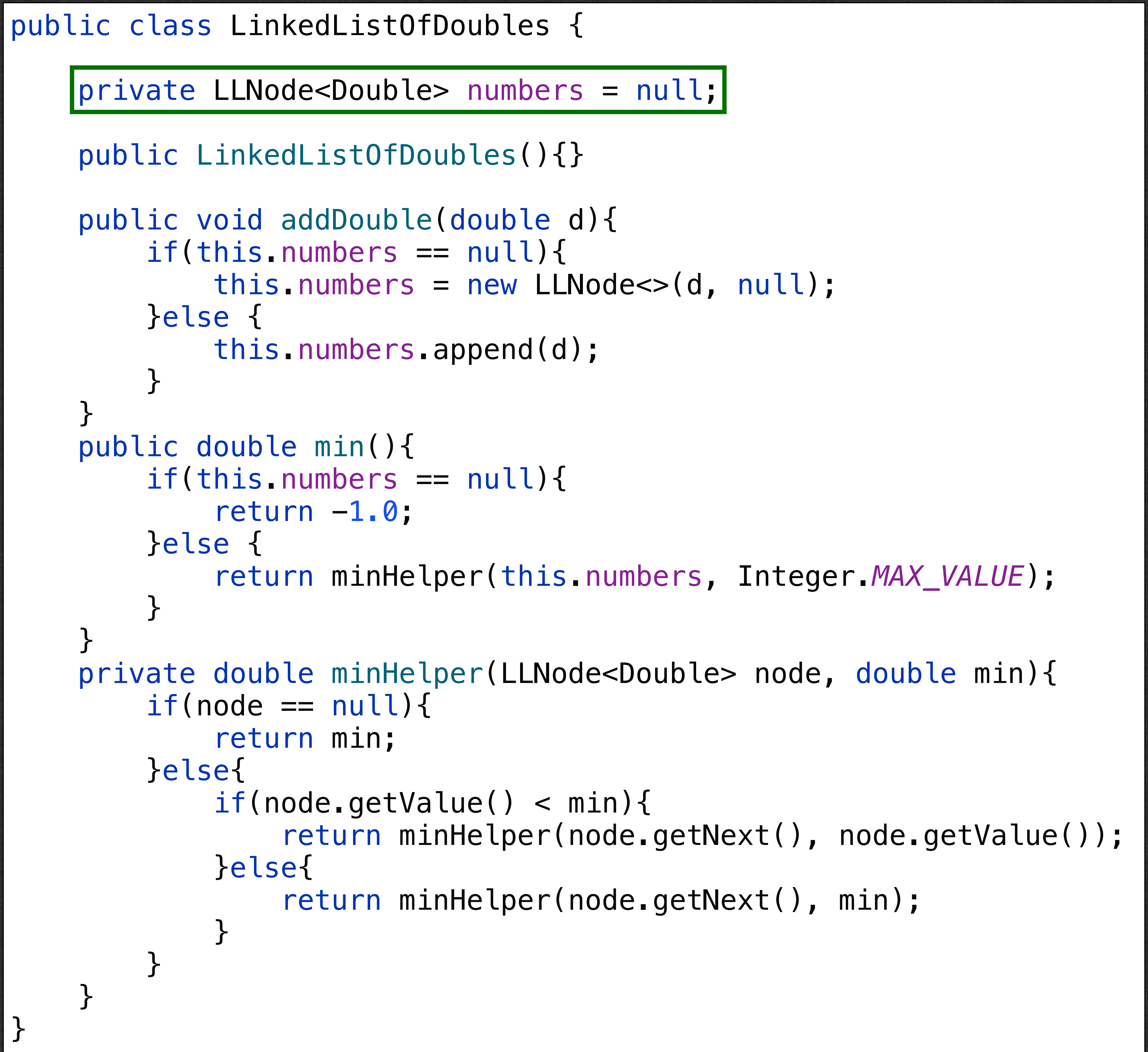
- The LinkedListOfDouble class contains a LinkedList of Doubles
  - Shortened to LLNode for the slide



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

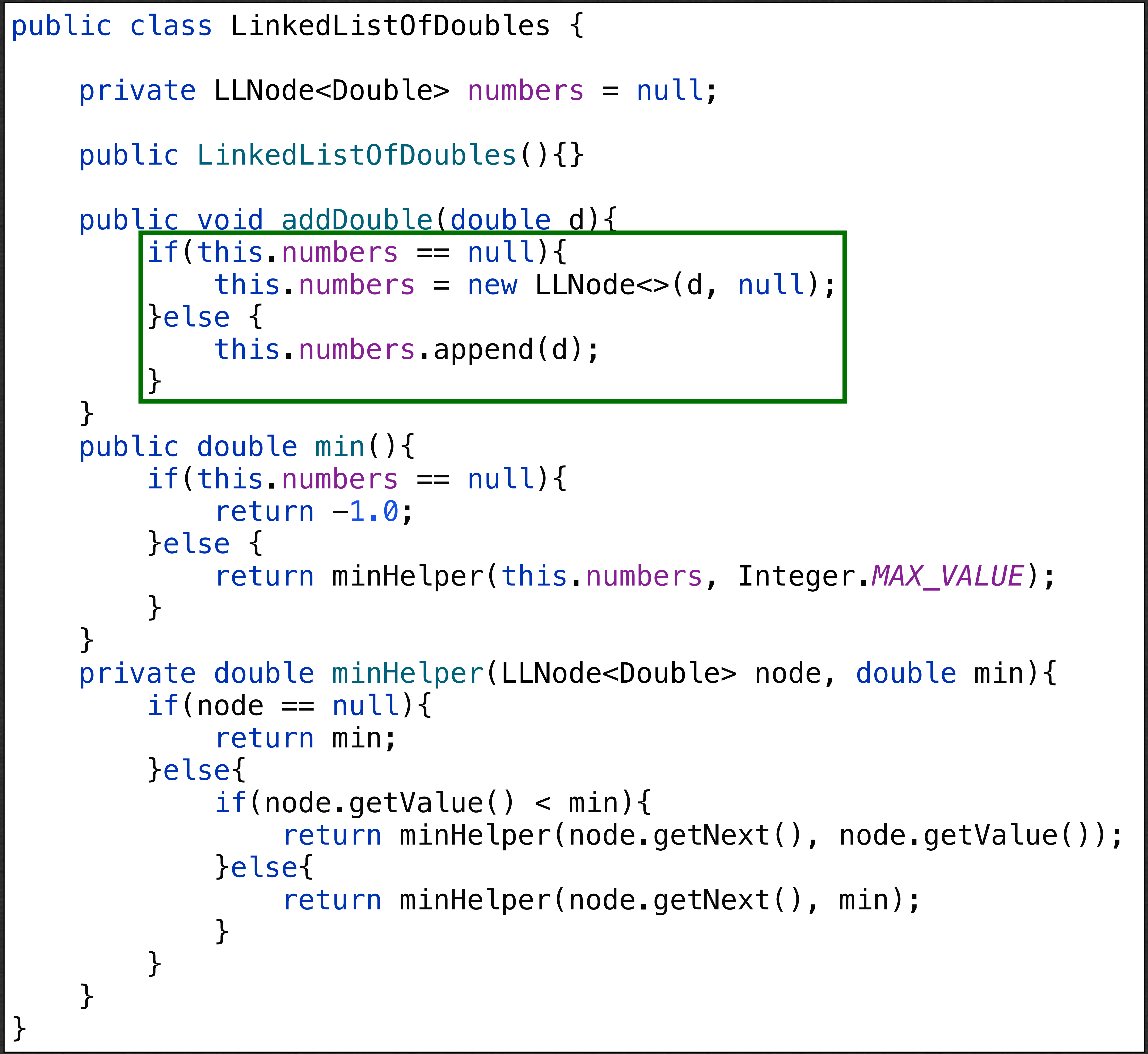
- When we create a Linked List variable that stores an empty list:
  - Set it to null!
  - Do NOT create a new Linked List Node since that would be a List of size 1
- Whenever working with the list, check if it's null
  - If it's null, it's empty



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- When adding a value to Linked List:
- Check if the list is null
    - If it is, it's empty
    - Create a new node to make a list of size 1
- If the list is not null
    - Add the new element to the existing list

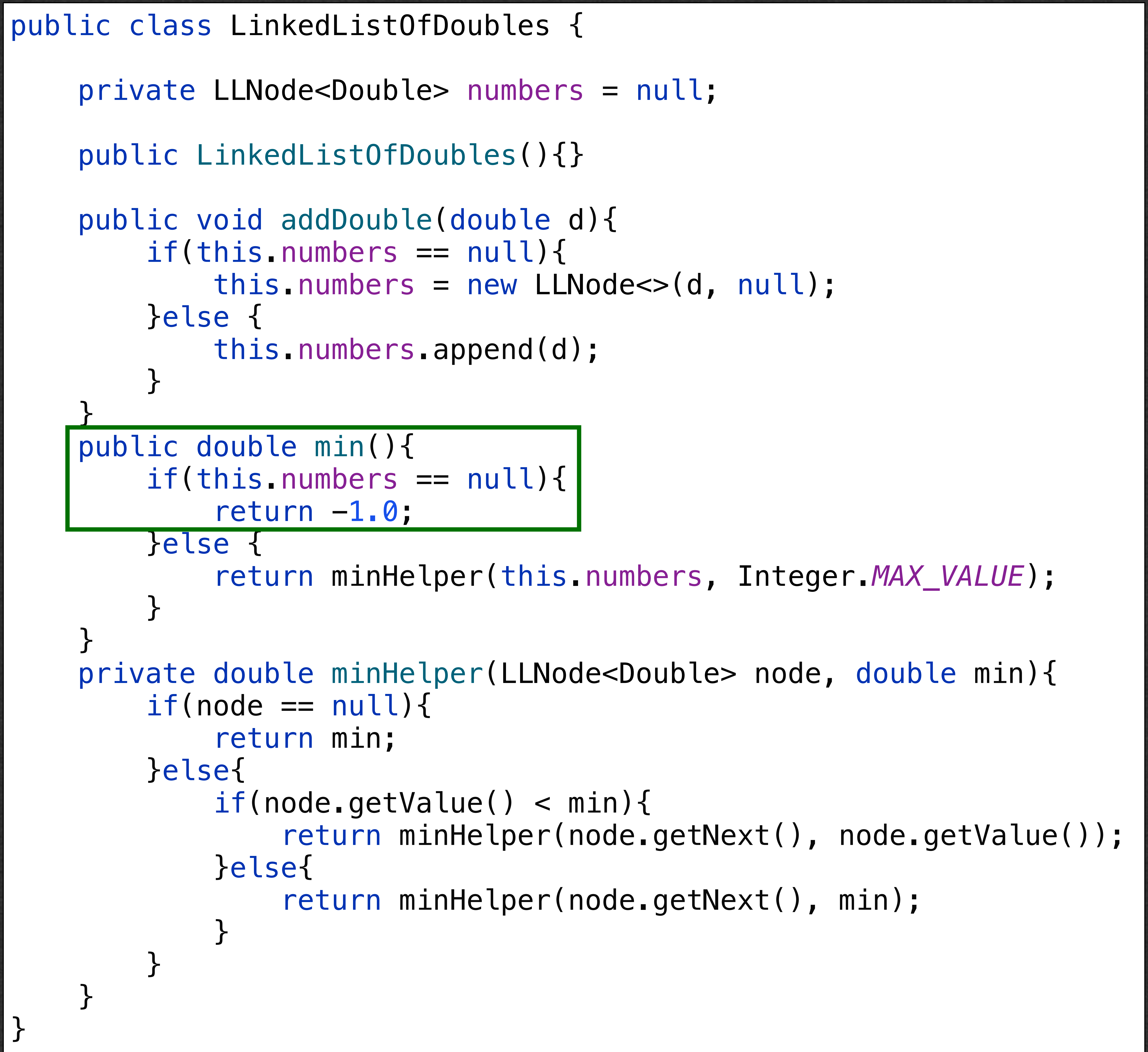


```
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- We want to write a min method that returns the min value in the List

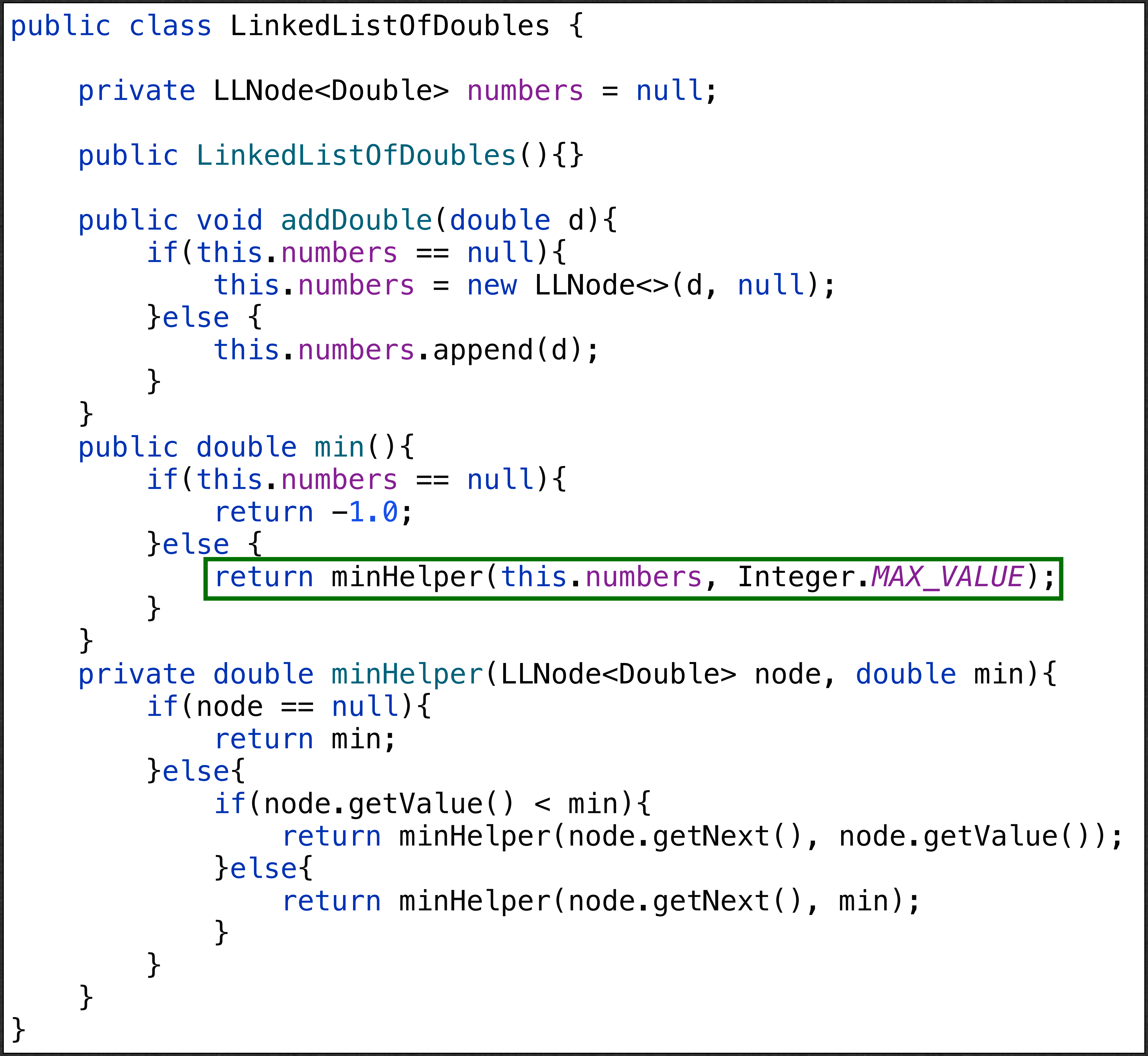
- First, if the List is empty we'll return -1.0 to indicate an error
    - *Only doing this for the example. This does introduce a bug where we can't tell if the min is actually -1.0



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

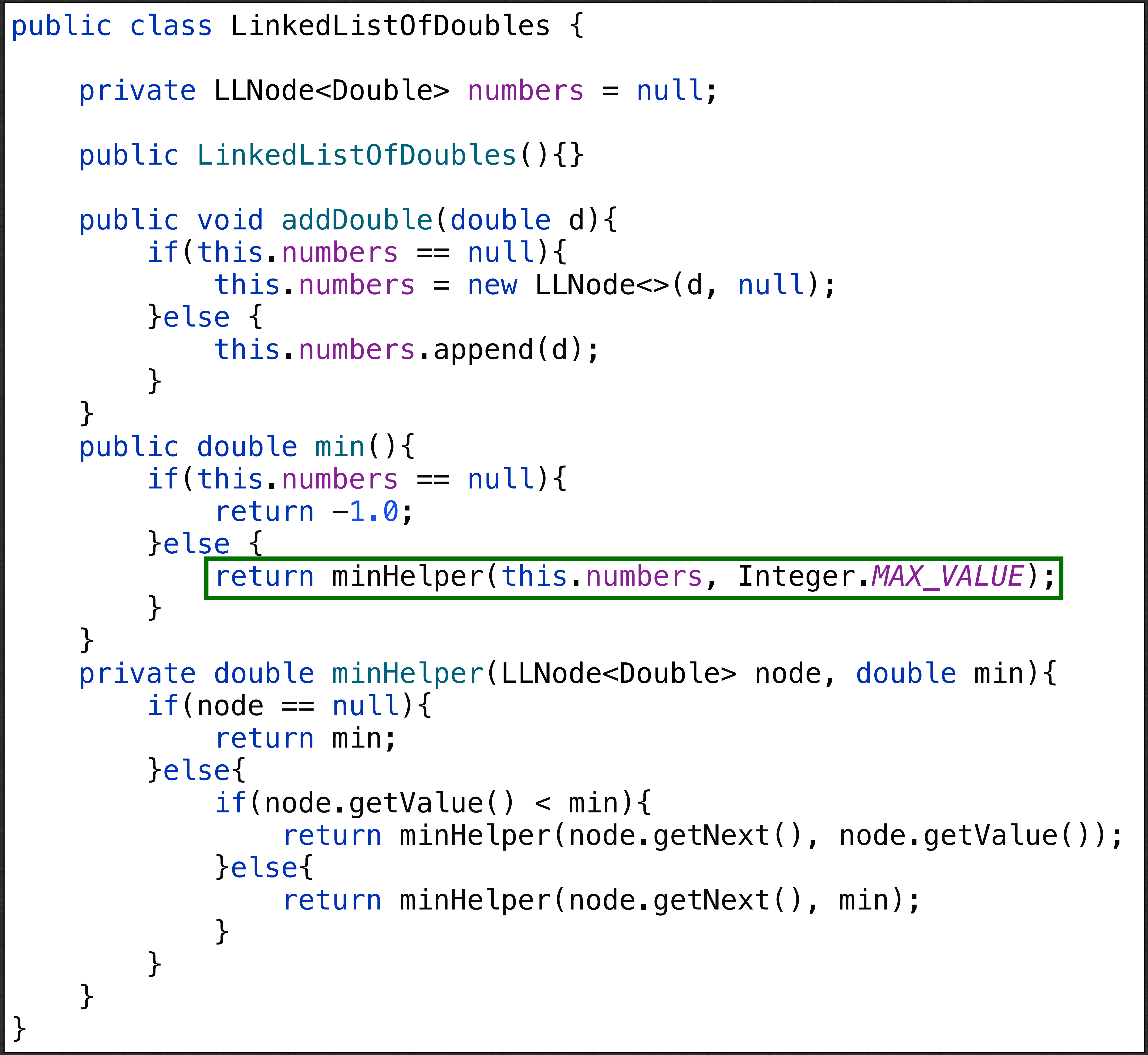
- We'd like to start the recursion..
- But min takes no parameters
    - We'd like to keep track of the min value through the recursive calls



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- We also need to track which node we're currently visiting
- In the previous examples, the code was in the LinkedListNode class
    - We had access to each node using "this"
- "this" is now a reference to the LinkedListOfDoubles (Not a LinkedListNode)



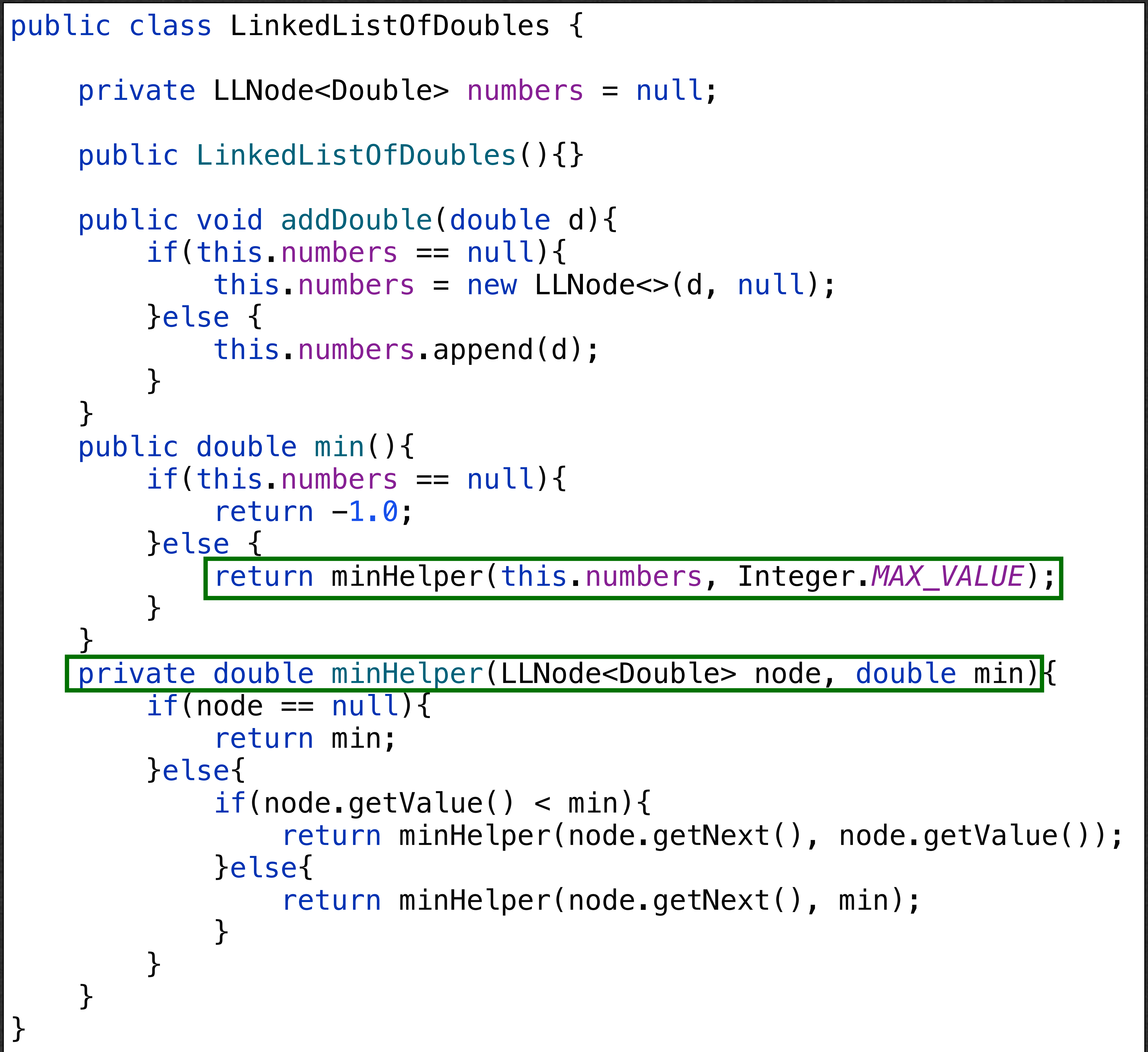
```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- Solution: Write a helper method to setup the recursion

- Add any parameters you want to help your recursive calls

- This helper takes a reference to the node being visited and a minimum value found so far

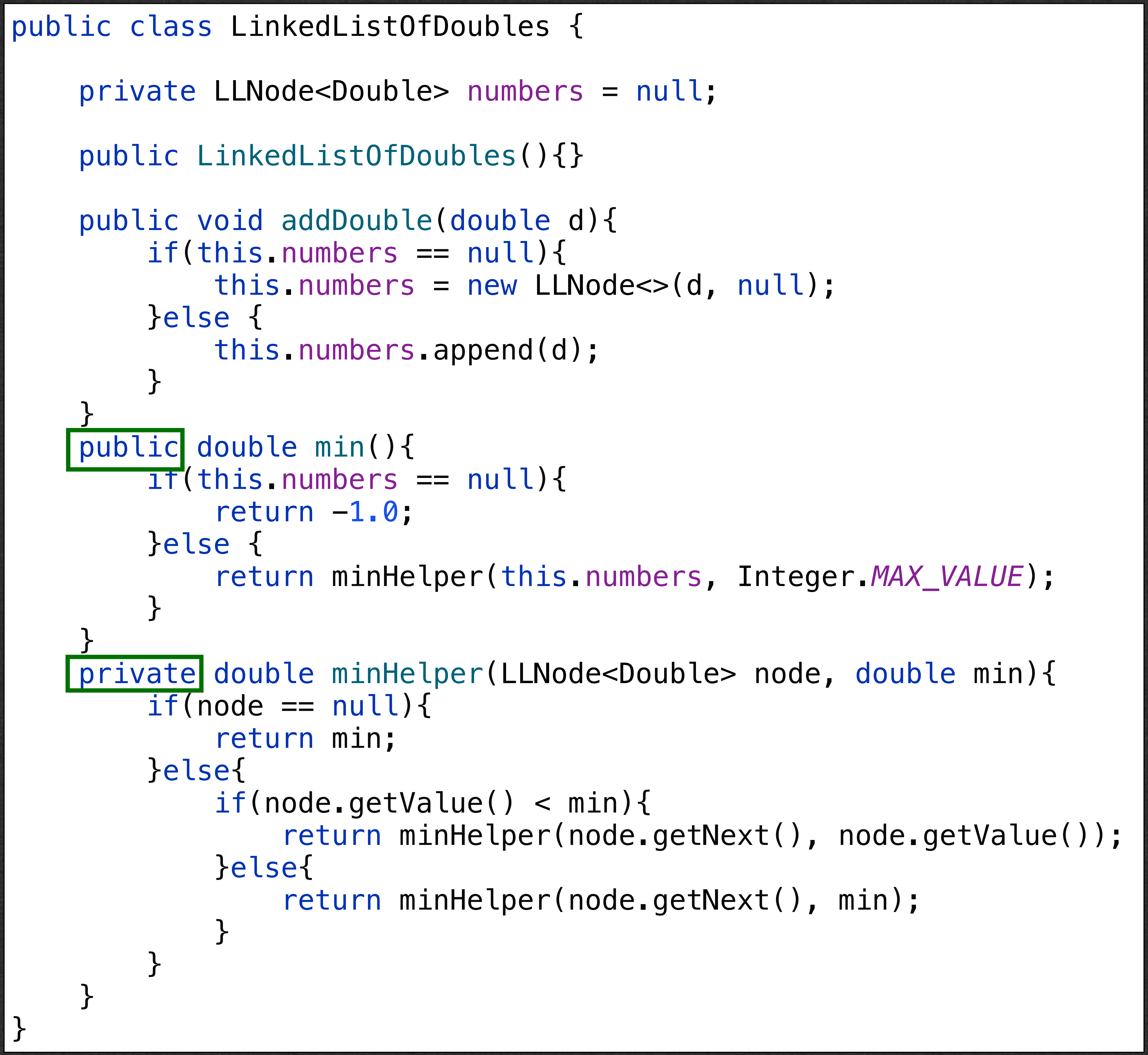


```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- We have a public method that people will call

- We have a private helper method that is a detail internal to this class
    - Anyone calling min does not care that this helper method exists
    - Make it private to hide the details (Encapsulation)

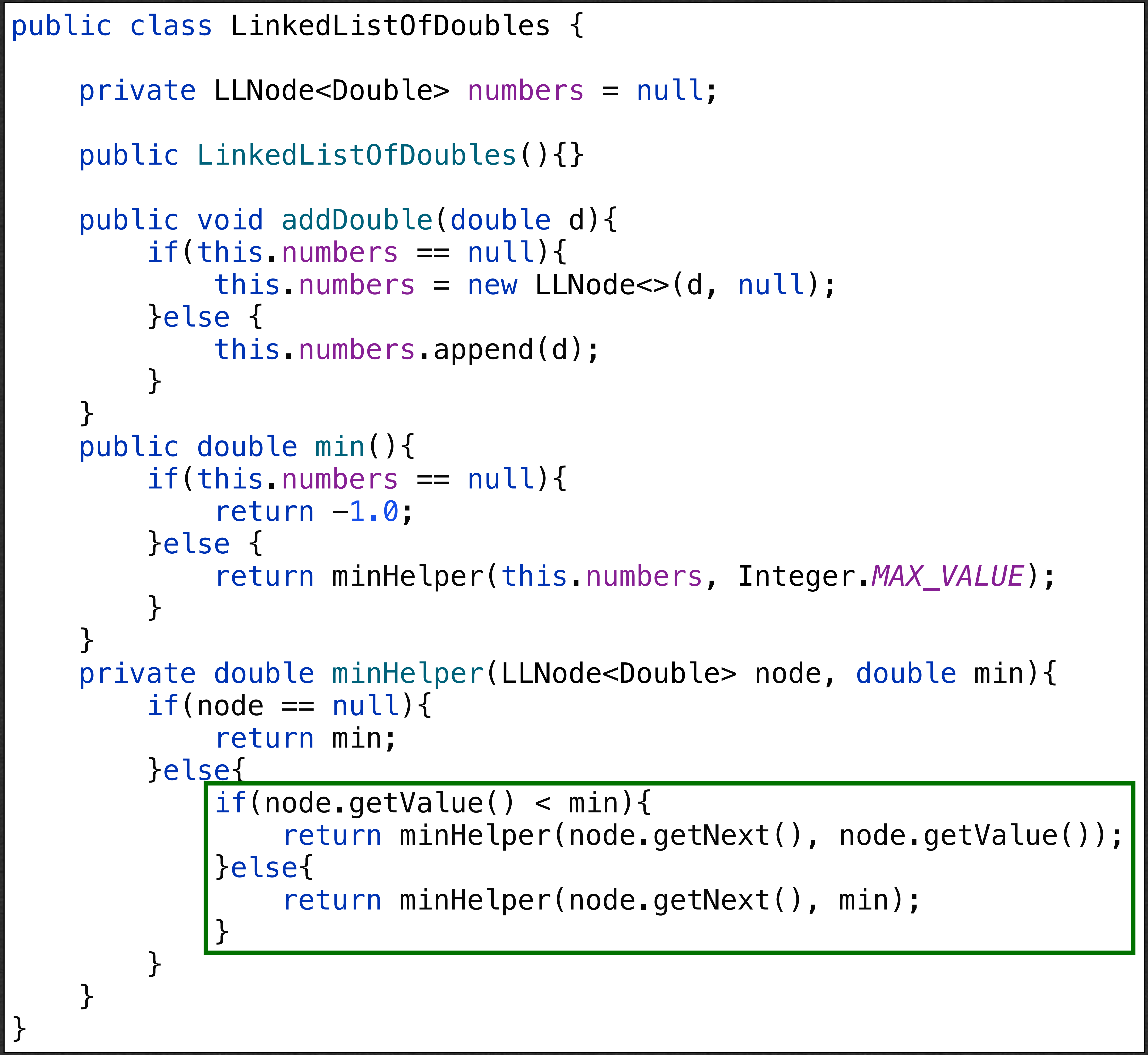


```
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

- Each recursive call is called with the next node in the list and the current min value

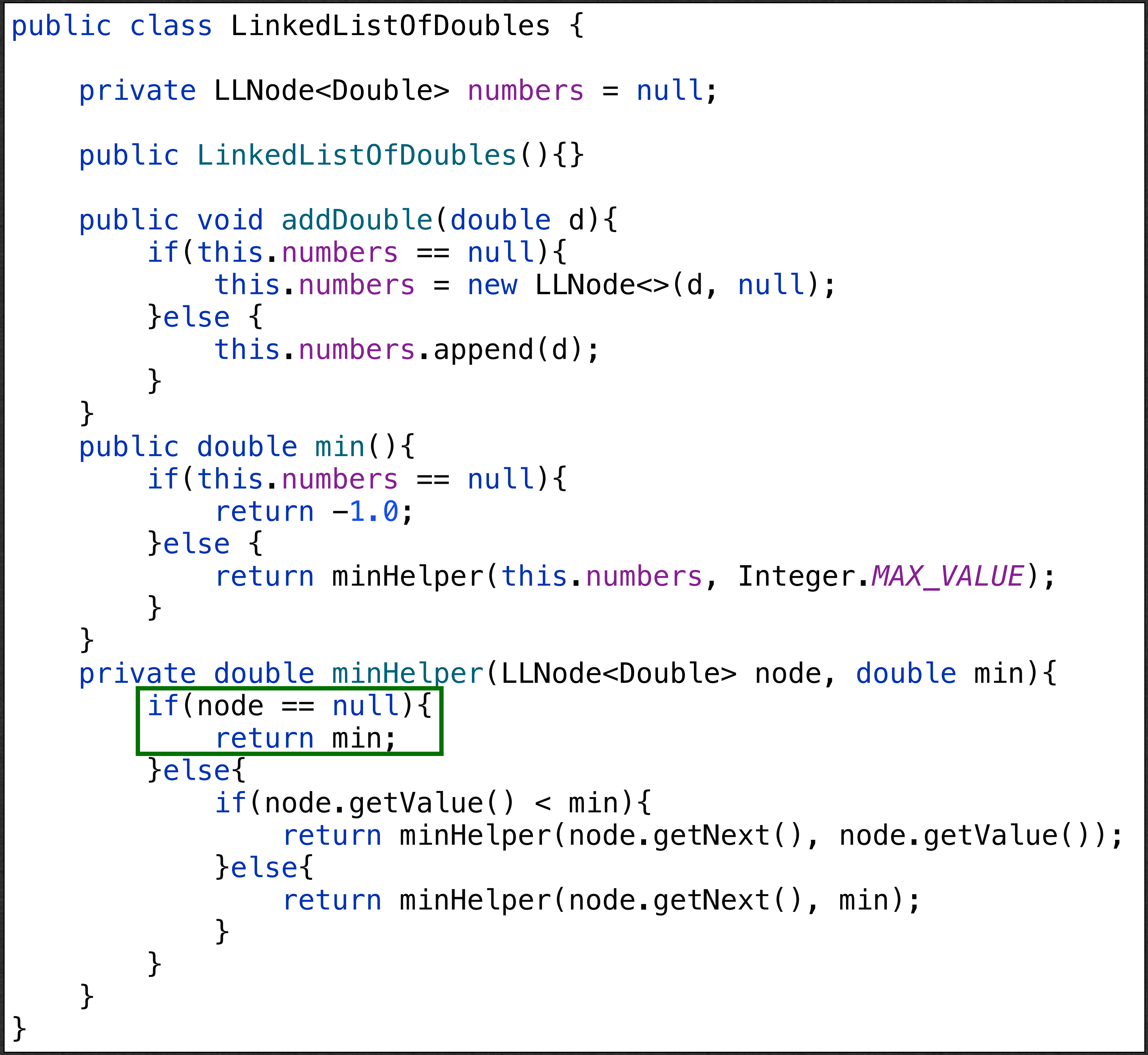
- If a node has a smaller value than min, update min for the next recursive call



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

Linked List - Min

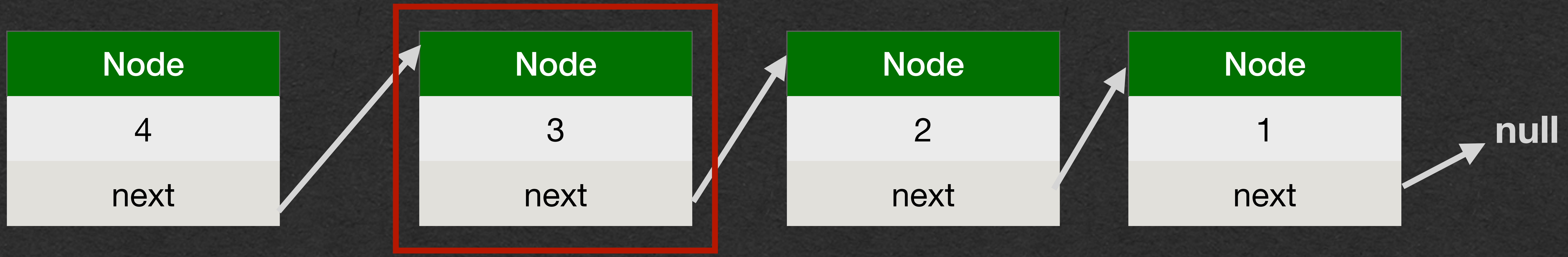
- When we reach the end of the list, return the min value
- At this point, all nodes have been checked so this is the final min value
- Return this value back up the recursive calls



```java
public class LinkedListOfDoubles {

    private LLNode<Double> numbers = null;

    public LinkedListOfDoubles(){}

    public void addDouble(double d){
        if(this.numbers == null){
            this.numbers = new LLNode<>(d, null);
        }else {
            this.numbers.append(d);
        }
    }
    public double min(){
        if(this.numbers == null){
            return -1.0;
        }else {
            return minHelper(this.numbers, Integer.MAX_VALUE);
        }
    }
    private double minHelper(LLNode<Double> node, double min){
        if(node == null){
            return min;
        }else{
            if(node.getValue() < min){
                return minHelper(node.getNext(), node.getValue());
            }else{
                return minHelper(node.getNext(), min);
            }
        }
    }
}
```

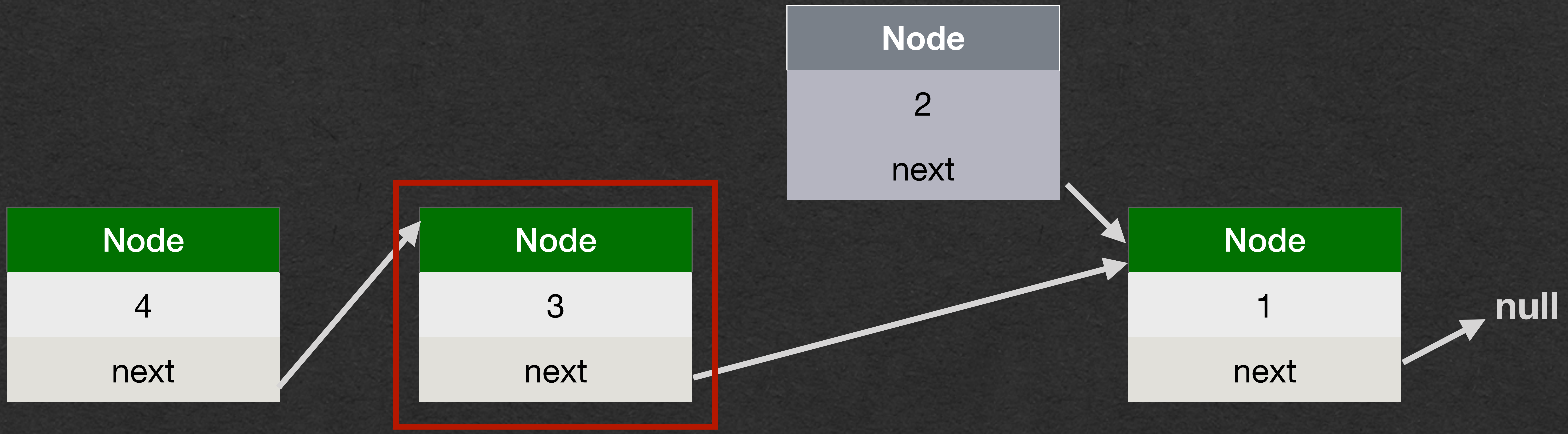
Delete a Node

- Want to delete the node containing 2
- Need a reference to the previous node



Delete a Node

- Update that node's next to bypass the deleted node
  - Don't have to update deleted node
  - The list no longer refers to this node


Node
2
next
Node
4
next
Node
3
next
Node
1
next
null