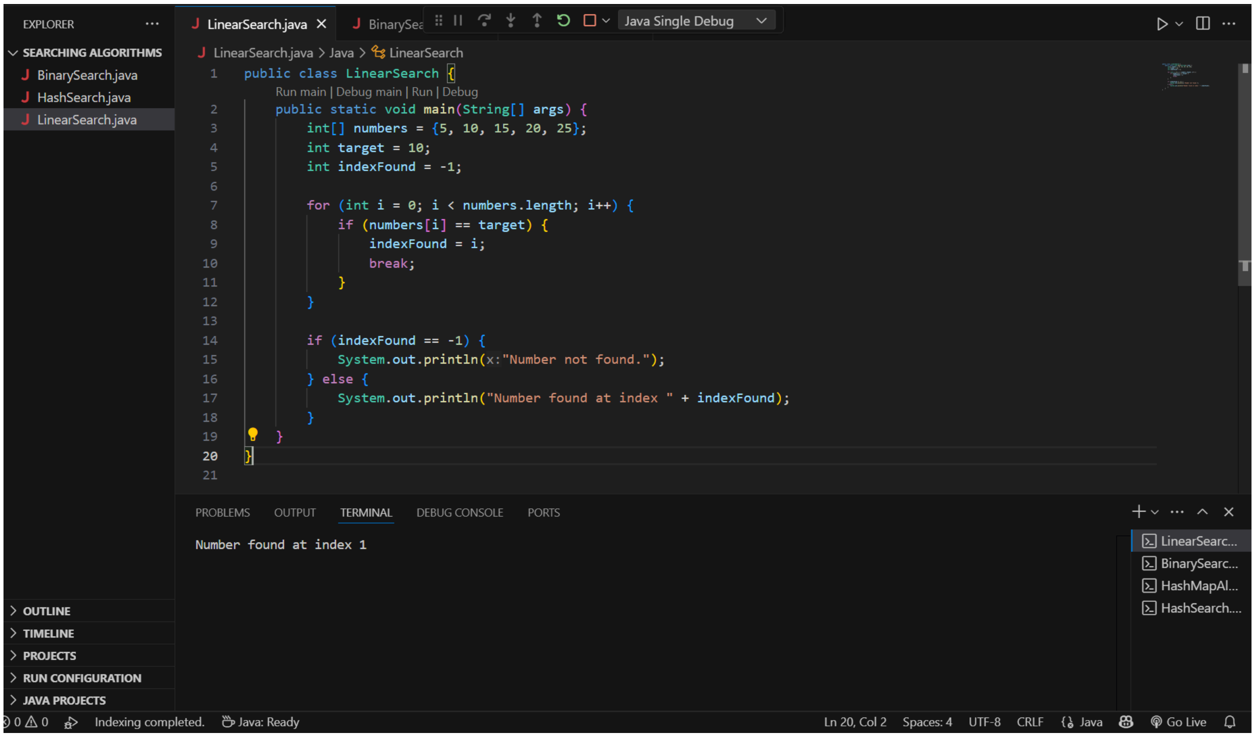
Linear Search



```java
public class LinearSearch {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        int target = 10;
        int indexFound = -1;

        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] == target) {
                indexFound = i;
                break;
            }
        }

        if (indexFound == -1) {
            System.out.println(x:"Number not found.");
        } else {
            System.out.println("Number found at index " + indexFound);
        }
    }
}
```

```
Number found at index 1
```

Explanation:

public class LinearSearch — Declares the class that contains the Linear Search logic.

public static void main(String[] args) —The main method where the program starts.

int[] numbers = {5, 10, 15, 20, 25}; — A sorted list of numbers to search through.

int target = 10; — The number we want to find in the array.

int indexFound = -1; — Stores the index where the number is found (starts as -1, which means "not found").

for (int i = 0; i < numbers.length; i++) — Loops through each element in the array.

if (numbers[i] == target) — Checks if the current number matches the target.

indexFound = i; — Saves the index where the number was found.

break; — Stops the loop since the number is already found.

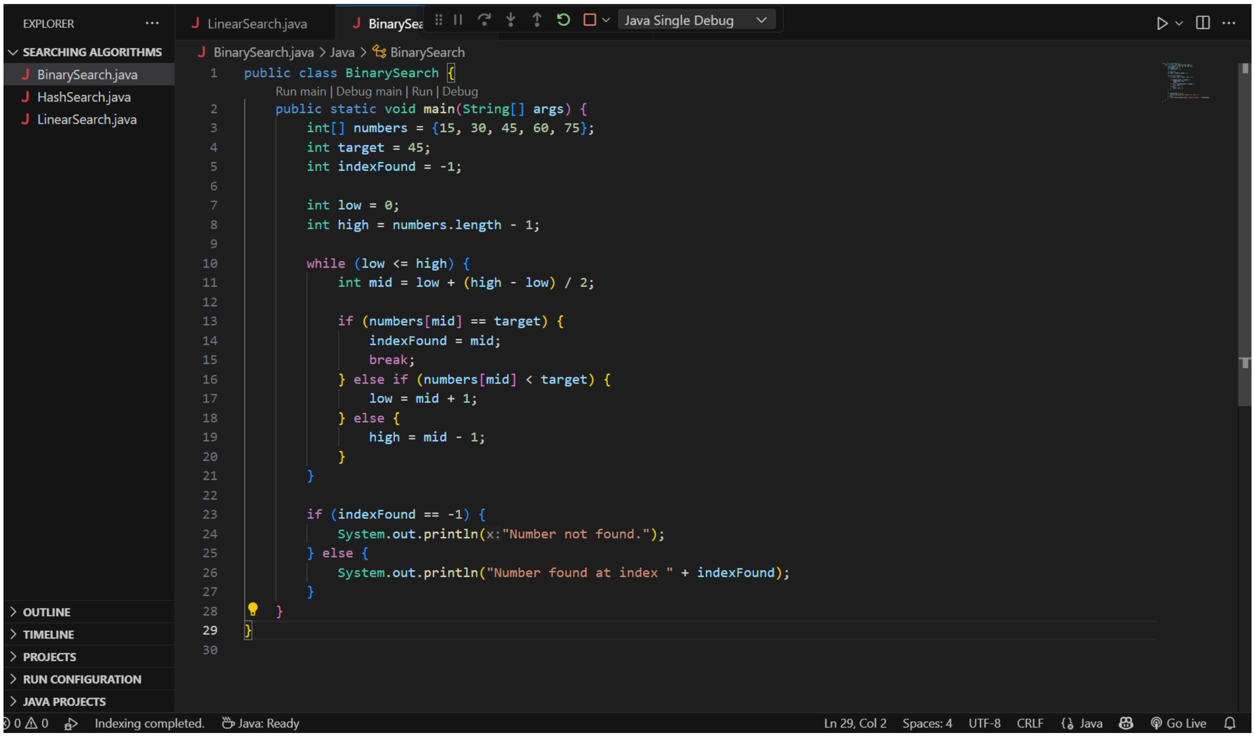
if (indexFound == -1) — Checks if the number was not found (still -1).

System.out.println("Number not found."); — Prints a message if the number wasn't found.

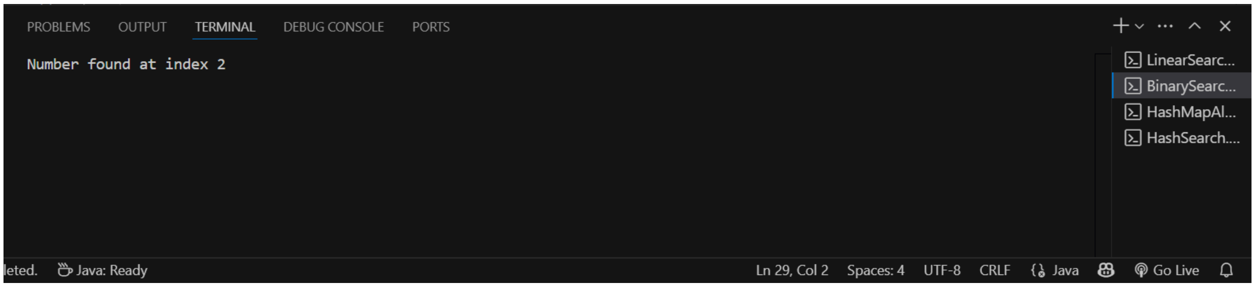
else — If the number was found.

System.out.println("Number found at index " + indexFound); — Prints the index where it was found.

Binary Search



```java
public class BinarySearch {
    public static void main(String[] args) {
        int[] numbers = {15, 30, 45, 60, 75};
        int target = 45;
        int indexFound = -1;

        int low = 0;
        int high = numbers.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (numbers[mid] == target) {
                indexFound = mid;
                break;
            } else if (numbers[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        if (indexFound == -1) {
            System.out.println(x:"Number not found.");
        } else {
            System.out.println("Number found at index " + indexFound);
        }
    }
}
```



```
Number found at index 2
```

Explanation:

public class BinarySearch — Declares the class that contains the Binary Search logic.

public static void main(String[] args) — The main method where the program starts.

int[] numbers = {15, 30, 45, 60, 75}; — A sorted list of numbers to search through.

int target = 45; — The number we want to find in the array.

int indexFound = -1; — Stores the index where the number is found (starts as -1, which means "not found").

int low = 0; — Sets the starting index of the search range.

int high = numbers.length - 1; — Sets the ending index of the search range.

while (low <= high) — Loop runs while there's a valid search range.

int mid = low + (high - low) / 2; — Calculates the middle index of the current range.

if (numbers[mid] == target) — Checks if the middle value is equal to the target.

indexFound = mid; — Saves the index where the target is found.

break ; — Stops the loop since the number is already found.

else if (numbers[mid] < target) — If the middle value is less than the target.

low = mid + 1; — narrow the search to the right half.

else — Otherwise, the middle value is greater than the target.

high = mid - 1; — so search the left half.

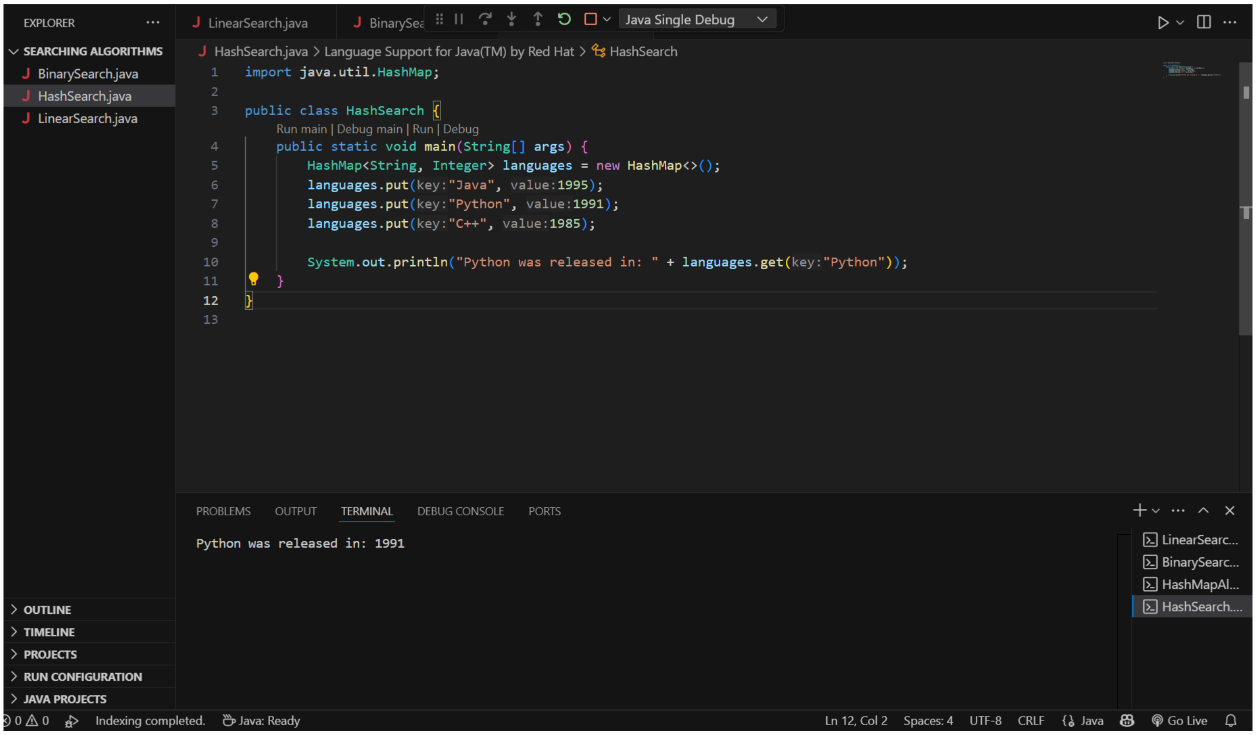
if (indexFound == -1) — Checks if the number was not found (still -1).

System.out.println("Number not found."); — Prints a message if the number wasn't found.

else — If the number was found.

System.out.println("Number found at index " + indexFound); — Prints the index where it was found.

Hashing



```java
import java.util.HashMap;

public class HashSearch {
    public static void main(String[] args) {
        HashMap<String, Integer> languages = new HashMap<>();
        languages.put(key:"Java", value:1995);
        languages.put(key:"Python", value:1991);
        languages.put(key:"C++", value:1985);

        System.out.println("Python was released in: " + languages.get(key:"Python"));
    }
}
```

```
Python was released in: 1991
```

Explanation:

import java.util.HashMap; — Imports the HashMap class.

public class HashSearch — Declares the class that contains the Hash Search logic.

public static void main(String[] args) — The main method where the program starts.

HashMap<String, Integer> languages = new HashMap<>(); — Creates a HashMap that maps programming language names (String) to their release years (Integer).

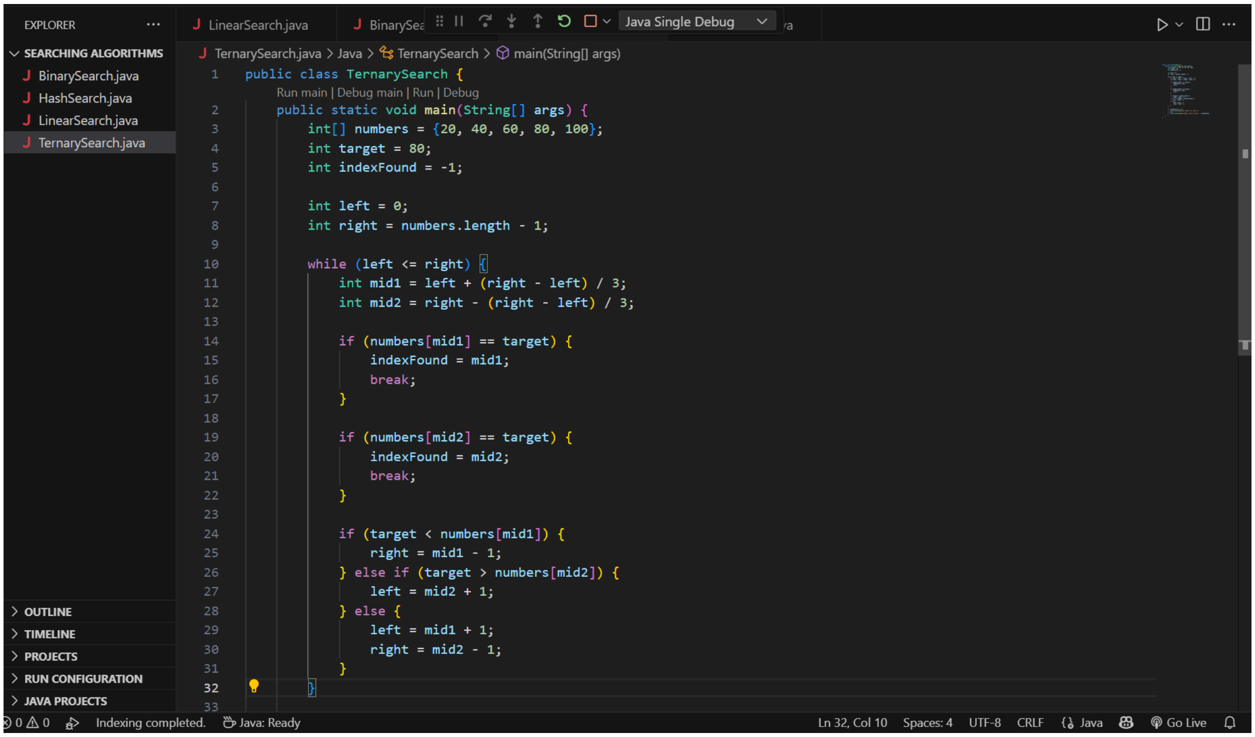
languages.put("Java", 1995); — Adds "Java" as a key with the value 1995.

languages.put("Python", 1991); — Adds "Python" as a key with the value 1991.

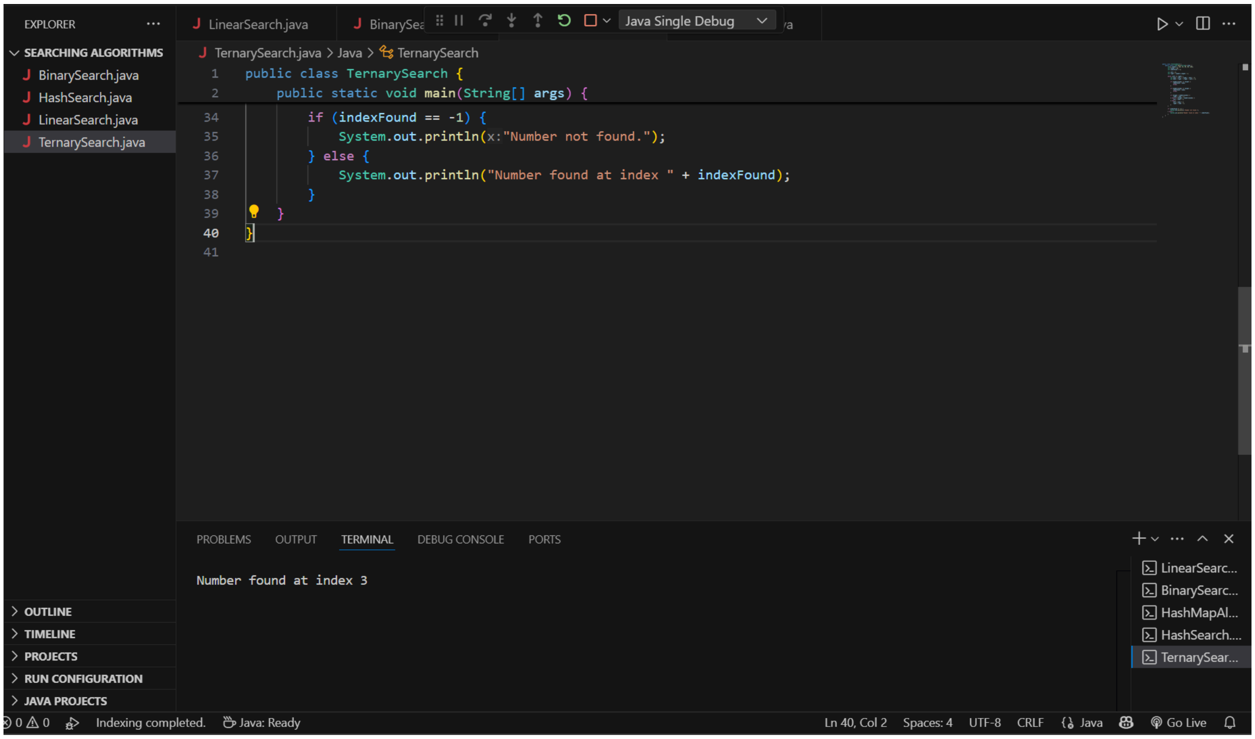
languages.put("C++", 1985); — Adds "C++" as a key with the value 1985.

System.out.println("Python was released in: " + languages.get("Python")); — Retrieves the release year of "Python" and prints it.

Ternary Search



```java
public class TernarySearch {
    public static void main(String[] args) {
        int[] numbers = {20, 40, 60, 80, 100};
        int target = 80;
        int indexFound = -1;

        int left = 0;
        int right = numbers.length - 1;

        while (left <= right) {
            int mid1 = left + (right - left) / 3;
            int mid2 = right - (right - left) / 3;

            if (numbers[mid1] == target) {
                indexFound = mid1;
                break;
            }

            if (numbers[mid2] == target) {
                indexFound = mid2;
                break;
            }

            if (target < numbers[mid1]) {
                right = mid1 - 1;
            } else if (target > numbers[mid2]) {
                left = mid2 + 1;
            } else {
                left = mid1 + 1;
                right = mid2 - 1;
            }
        }

```



```java
public class TernarySearch {
    public static void main(String[] args) {
        if (indexFound == -1) {
            System.out.println(x:"Number not found.");
        } else {
            System.out.println("Number found at index " + indexFound);
        }
    }
}

```

```
Number found at index 3
```

Explanation:

public class TernarySearch — Declares the class that contains the Ternary Search logic.

public static void main(String[] args) — The main method where the program starts.

int[] numbers = {20, 40, 60, 80, 100}; — A sorted list of numbers to search through.

int target = 80; — The number we want to find in the array.

int indexFound = -1; — Stores the index where the number is found (starts as -1, which means "not found").

int left = 0; — Sets the starting index of the search range.

int right = numbers.length - 1; — Sets the ending index of the search range.

while (left <= right) — Loop runs while there’s a valid range to search in.

int mid1 = left + (right - left) / 3; — Calculates the first third index.

int mid2 = right - (right - left) / 3; — Calculates the second third index.

if (numbers[mid1] == target) — Checks if the target is at the first third.

indexFound = mid1; — Saves the index if found at mid1.

break; — Stops the loop since the number is already found.

if (numbers[mid2] == target) — Checks if the target is at the second third.

indexFound = mid2; — Saves the index if found at mid2.

break; — Stops the loop since the number is already found.

if (target < numbers[mid1]) — If the target is less than the first third.

right = mid1 - 1; — Search in the first third.

else if (target > numbers[mid2]) — If the target is greater than the second third.

left = mid2 + 1; — Search in the last third.

else — Otherwise, the target is between mid1 and mid2.

left = mid1 + 1; — Search in the middle third (start from just after mid1).

right = mid2 - 1; — End the search at just before mid2.

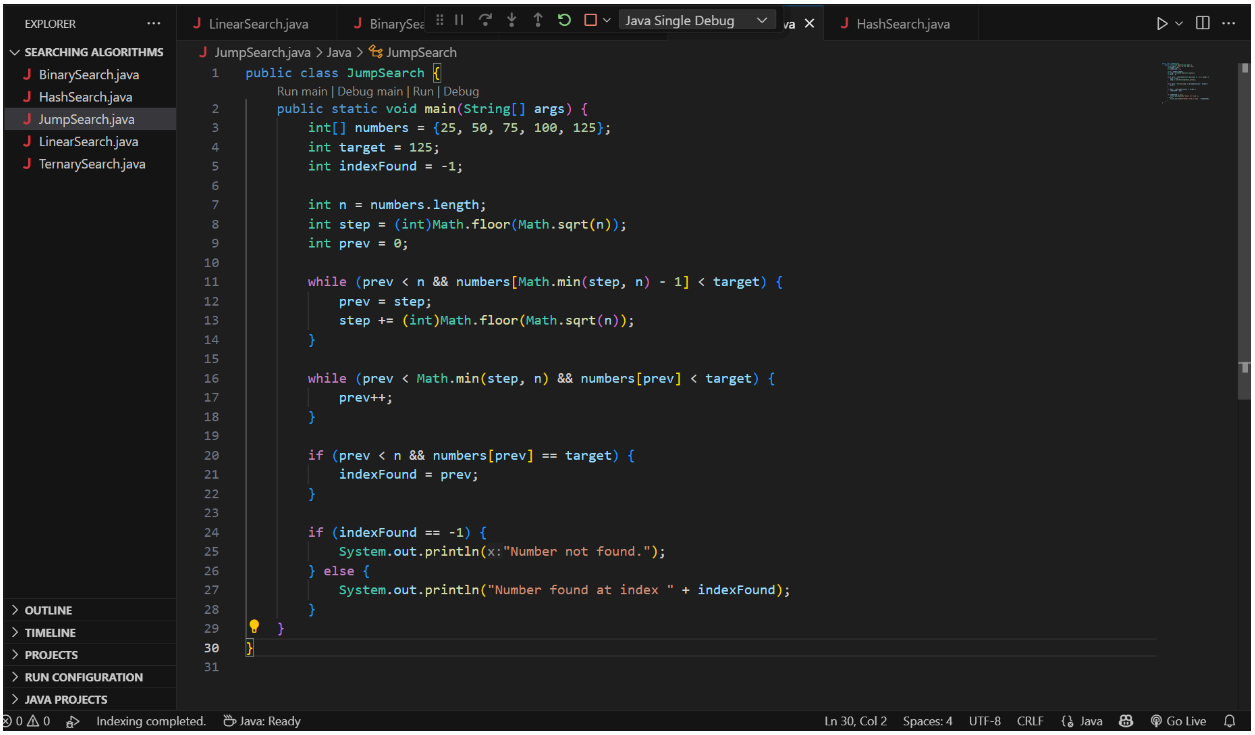
if (indexFound == -1) — Checks if the number was not found (still -1).

System.out.println("Number not found."); — Prints a message if the number wasn't found.

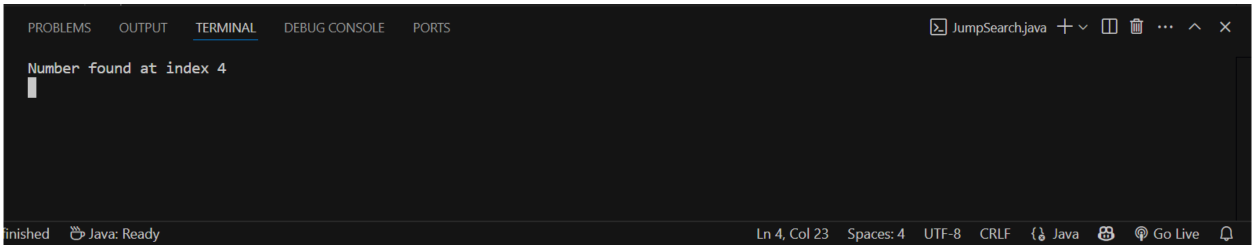
else — If the number was found.

System.out.println("Number found at index " + indexFound); — Prints the index where it was found.

Jump Search



```java
public class JumpSearch {
    public static void main(String[] args) {
        int[] numbers = {25, 50, 75, 100, 125};
        int target = 125;
        int indexFound = -1;

        int n = numbers.length;
        int step = (int)Math.floor(Math.sqrt(n));
        int prev = 0;

        while (prev < n && numbers[Math.min(step, n) - 1] < target) {
            prev = step;
            step += (int)Math.floor(Math.sqrt(n));
        }

        while (prev < Math.min(step, n) && numbers[prev] < target) {
            prev++;
        }

        if (prev < n && numbers[prev] == target) {
            indexFound = prev;
        }

        if (indexFound == -1) {
            System.out.println(x:"Number not found.");
        } else {
            System.out.println("Number found at index " + indexFound);
        }
    }
}
```



```
Number found at index 4
```

Explanation:

public class JumpSearch — Declares the class that contains the Jump Search logic.

public static void main(String[] args) — The main method where the program starts.

int[] numbers = {25, 50, 75, 100, 125}; — A sorted list of numbers to search through.

int target = 125; — The number we want to find in the array.

int indexFound = -1; — Stores the index where the number is found (starts as -1, which means "not found").

int n = numbers.length; — Gets the total number of elements in the array.

int step = (int)Math.floor(Math.sqrt(n)); — Calculates the jump size (√n) for block-wise searching.

int prev = 0; — Keeps track of the starting index of the current block.

while (prev < n && numbers[Math.min(step, n) - 1] < target) — Jump ahead in blocks while the end of the block is less than the target.

prev = step; — Move to the next block.

step += (int)Math.floor(Math.sqrt(n)); — Add the jump size again for the next block.

while (prev < Math.min(step, n) && numbers[prev] < target) — Do a linear search within the current block.

prev++; — Move to the next index in the block.

if (prev < n && numbers[prev] == target) — Check if the current element is the target.

indexFound = prev; — Save the index where the target was found.

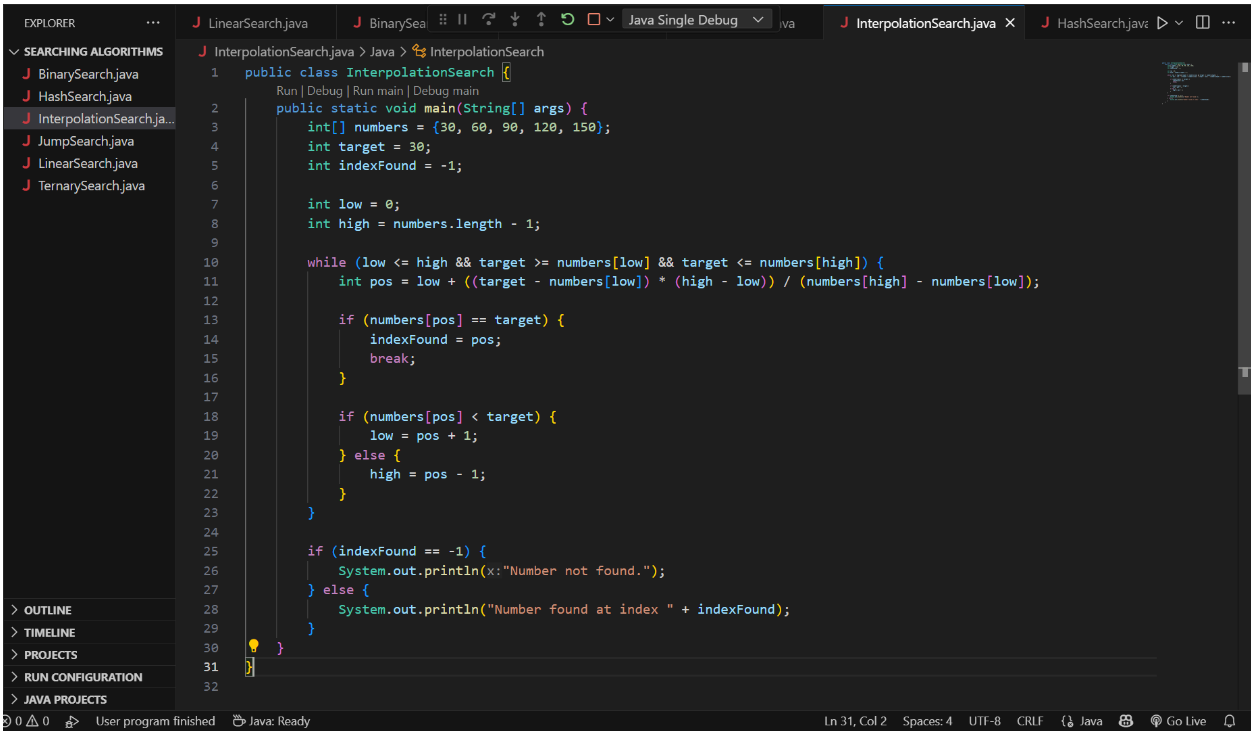
if (indexFound == -1) — Checks if the number was not found (still -1).

System.out.println("Number not found."); — Prints a message if the number wasn't found.

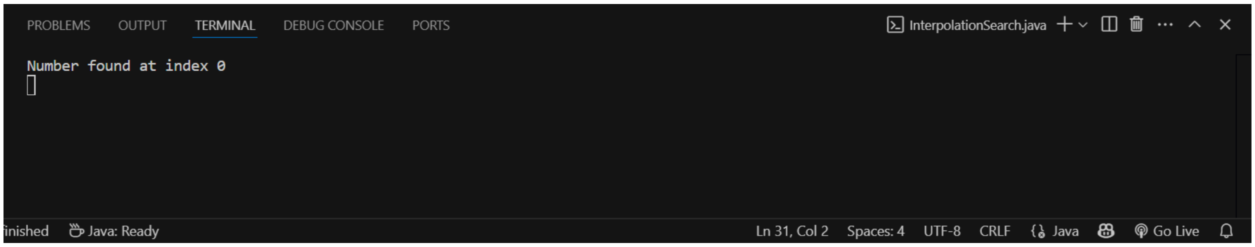
else — If the number was found.

System.out.println("Number found at index " + indexFound); — Prints the index where it was found.

Interpolation Search



```java
public class InterpolationSearch {
    public static void main(String[] args) {
        int[] numbers = {30, 60, 90, 120, 150};
        int target = 30;
        int indexFound = -1;

        int low = 0;
        int high = numbers.length - 1;

        while (low <= high && target >= numbers[low] && target <= numbers[high]) {
            int pos = low + ((target - numbers[low]) * (high - low)) / (numbers[high] - numbers[low]);

            if (numbers[pos] == target) {
                indexFound = pos;
                break;
            }

            if (numbers[pos] < target) {
                low = pos + 1;
            } else {
                high = pos - 1;
            }
        }

        if (indexFound == -1) {
            System.out.println(x:"Number not found.");
        } else {
            System.out.println("Number found at index " + indexFound);
        }
    }
}
```



```
Number found at index 0
```

Explanation:

public class InterpolationSearch — Declares the class that contains the Interpolation Search logic.

public static void main(String[] args) — The main method where the program starts.

int[] numbers = {30, 60, 90, 120, 150}; — A sorted list of numbers to search through.

int target = 30; — The number we want to find in the array.

int indexFound = -1; — Stores the index where the number is found (starts as -1, which means "not found").

int low = 0; — Starting index of the search range.

int high = numbers.length - 1; — Ending index of the search range.

while (low <= high && target >= numbers[low] && target <= numbers[high]) — Loop while the target is within the current search range.

int pos = low + ((target - numbers[low]) * (high - low)) / (numbers[high] - numbers[low]); — Estimate the position where the target might be, based on linear interpolation.

if (numbers[pos] == target) — Check if the estimated position contains the target.

indexFound = pos; — Save the position if target is found.

break; — Stops the loop since the number is already found.

if (numbers[pos] < target) — If the value at estimated position is less than target.

low = pos + 1; — Narrow search to the right half.

else — Otherwise, the value is greater than the target.

high = pos - 1; — Narrow search to the left half.

if (indexFound == -1) — Checks if the number was not found (still -1).

System.out.println("Number not found."); — Prints a message if the number wasn’t found.

else — If the number was found.

System.out.println("Number found at index " + indexFound); — Print the index where target was found.