Lab Assignment/Task Report

Only for course Teacher						
		Needs Improvement	Developing	Sufficient	Above Average	Total Mark
Allocate mark & Percentage		25%	50%	75%	100%	
Clarity						
Content Quality						
Spelling & Grammar						
Organization and Formatting						
					Total obtained mark	
Comments						

Semester: 2nd

Student Name: Rony Chowdhury Joy

Student ID:0022520005101024

Batch:45th **Section:A**

Course Code:CSE 124 **Course Name:OOP(Java)**

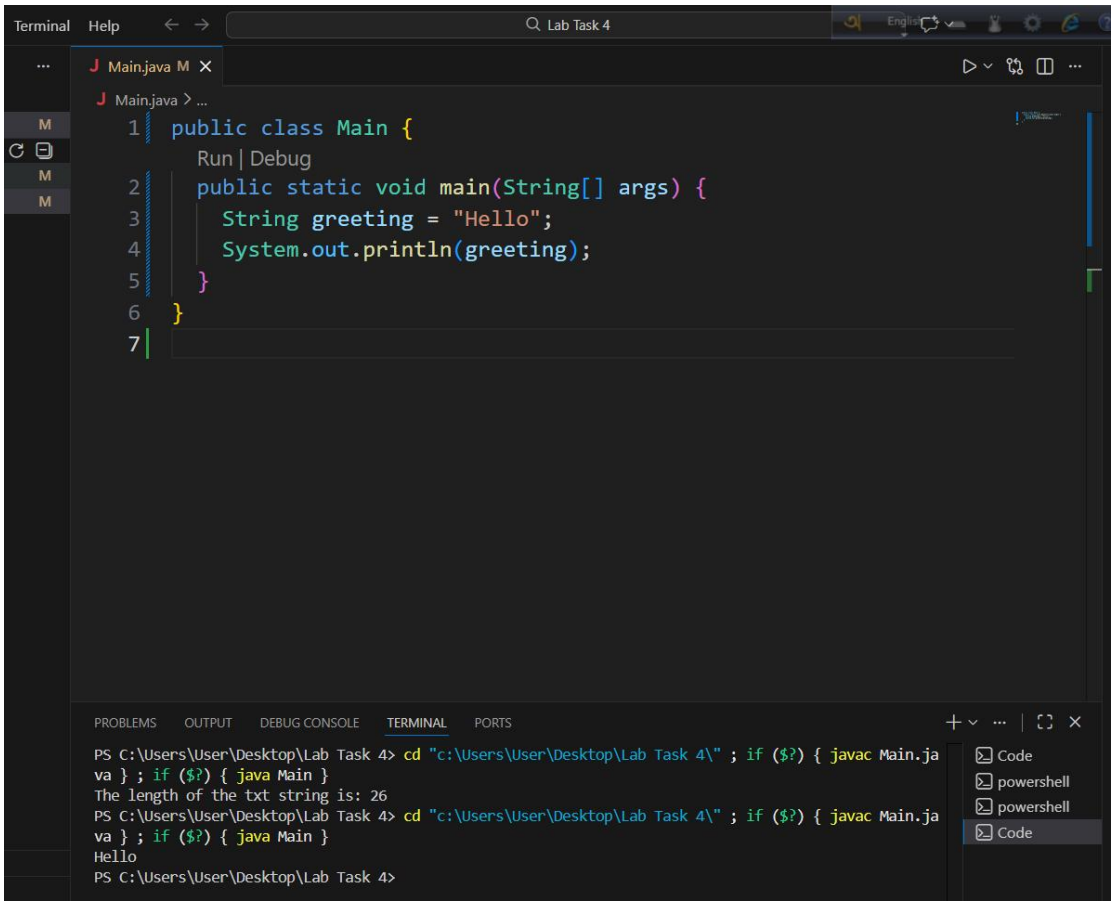
Course Teacher Name:Debabarata Mallick

Designation:Lecturer

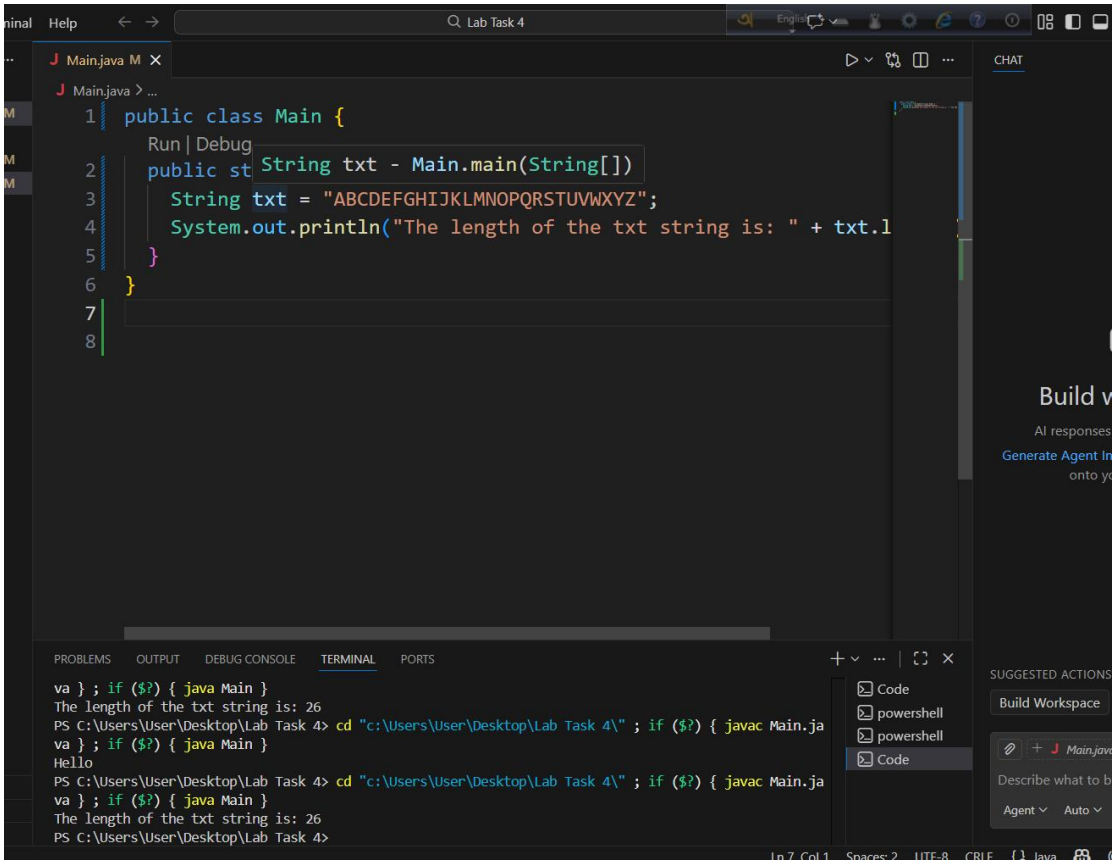
Submission Date:30/01/2026

Java Strings

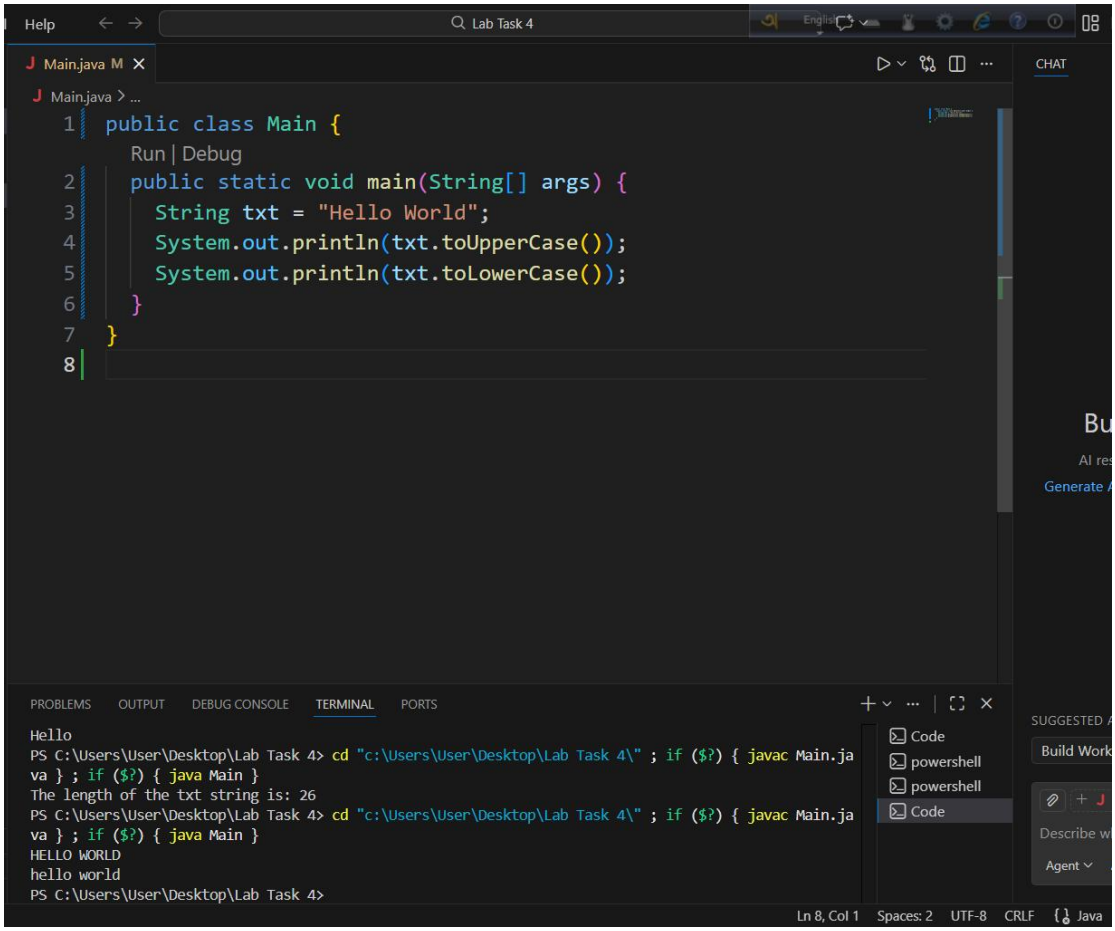
This program defines a class named Main. The main method is the starting point of the program where execution begins. Inside the main method, a String variable named greeting is created and assigned the value "Hello". Finally, the System.out.println(greeting); statement is used to display the value of the variable on the screen.



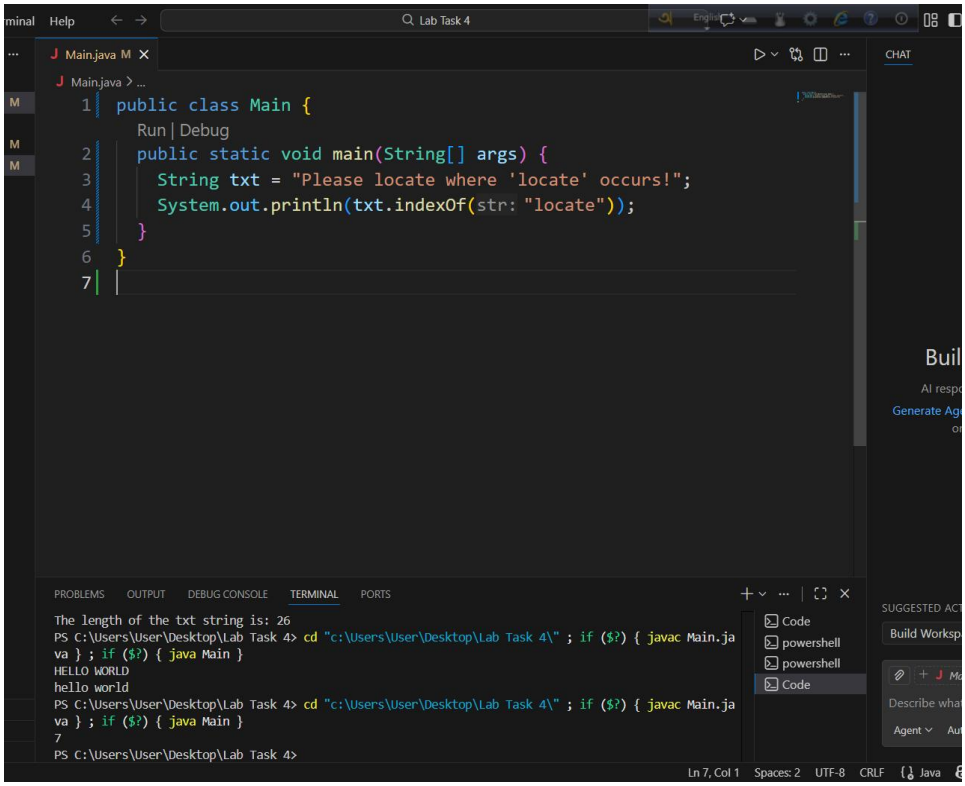
This program defines a class named Main, where the main method is the entry point of the program. Inside the main method, a String variable txt is created and assigned a sequence of alphabet letters. The length() method is used to find the total number of characters in the string. Finally, the result is printed on the screen using System.out.println, which shows the length of the string.



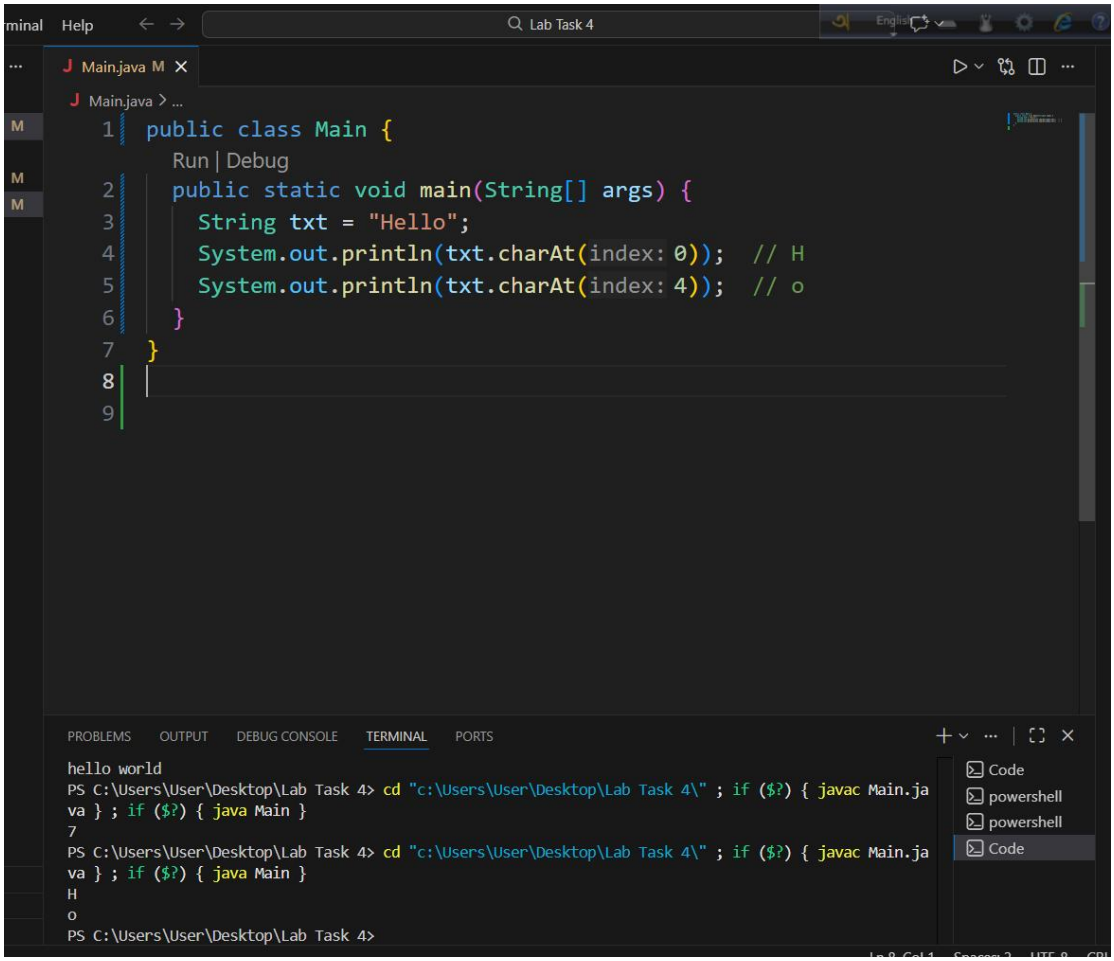
This program defines a class named Main, and the main method is where the program starts running. Inside the main method, a String variable txt is created and assigned the value "Hello World". The toUpperCase() method is used to convert all characters of the string into uppercase letters, and the toLowerCase() method is used to convert all characters into lowercase letters. Both converted strings are then printed on the screen using System.out.println.



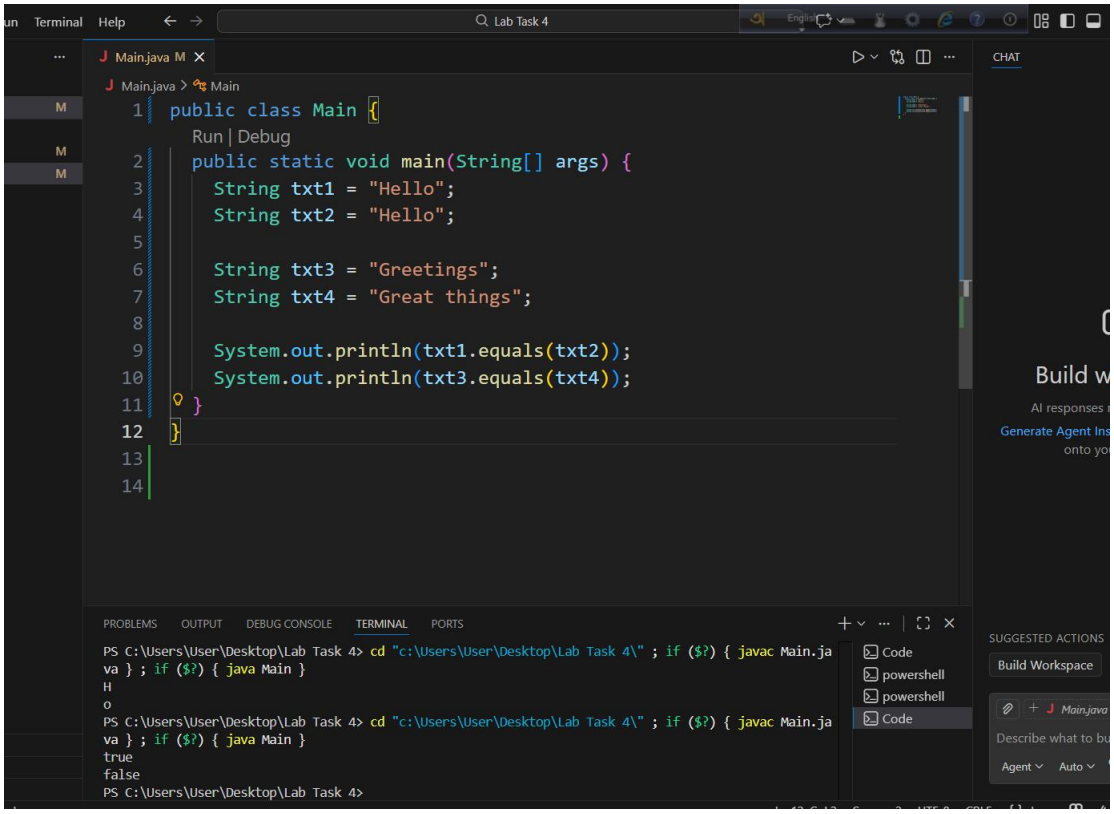
This program defines a class named Main, and the main method is the starting point of the program. Inside the main method, a String variable txt is created and assigned a sentence. The indexOf() method is used to find the position (index) of the word "locate" within the string. Finally, the index number of the first occurrence of the word is printed on the screen using System.out.println.



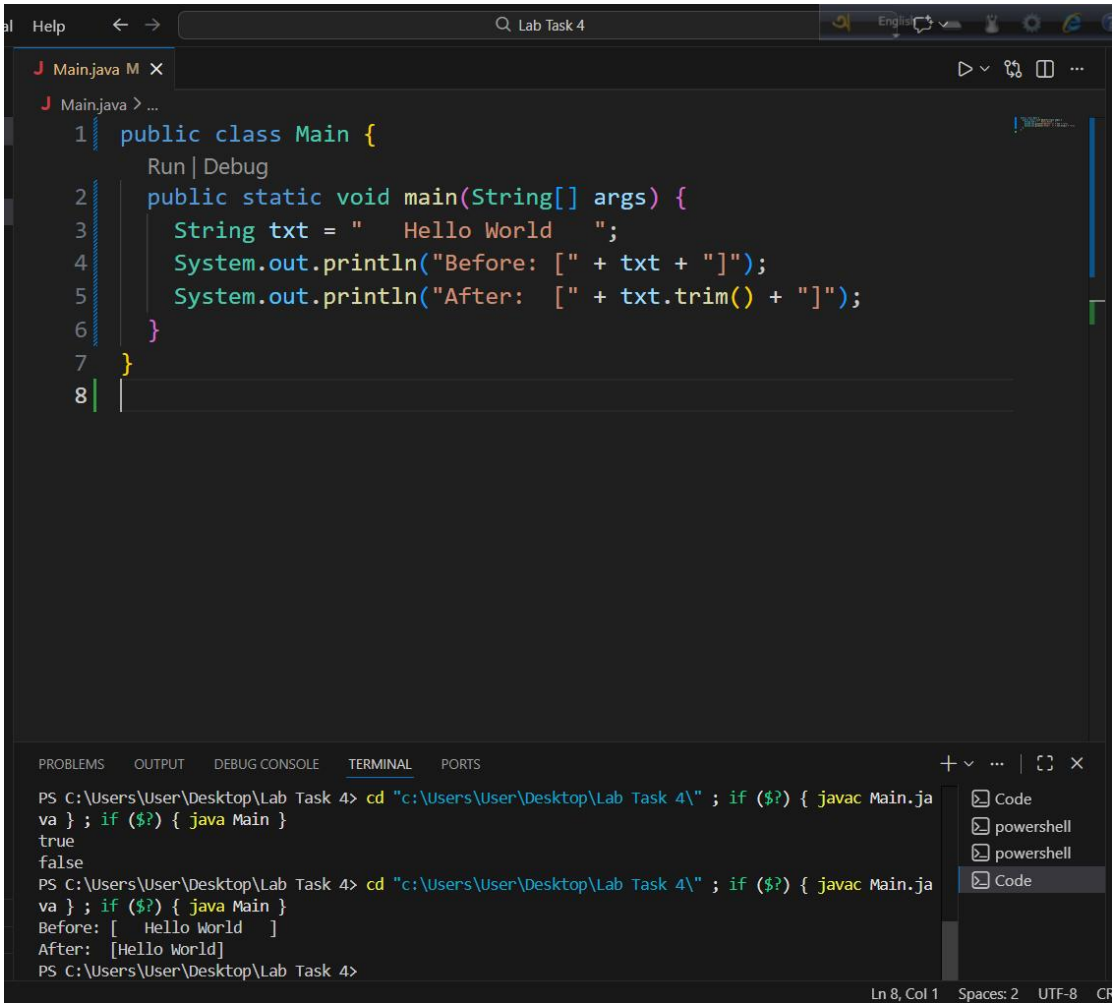
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a String variable txt is created and assigned the value "Hello". The charAt() method is used to access a specific character of the string by its index number. Here, index 0 prints the first character H, and index 4 prints the last character o. Both characters are displayed on the screen using System.out.println.



This program defines a class named Main, and the main method is the starting point of the program. Inside the main method, two String variables txt1 and txt2 are created with the same value "Hello", and two other String variables txt3 and txt4 are created with different values. The equals() method is used to compare the content of two strings. If the strings are equal, it returns true; otherwise, it returns false. Therefore, the first output is true and the second output is false.

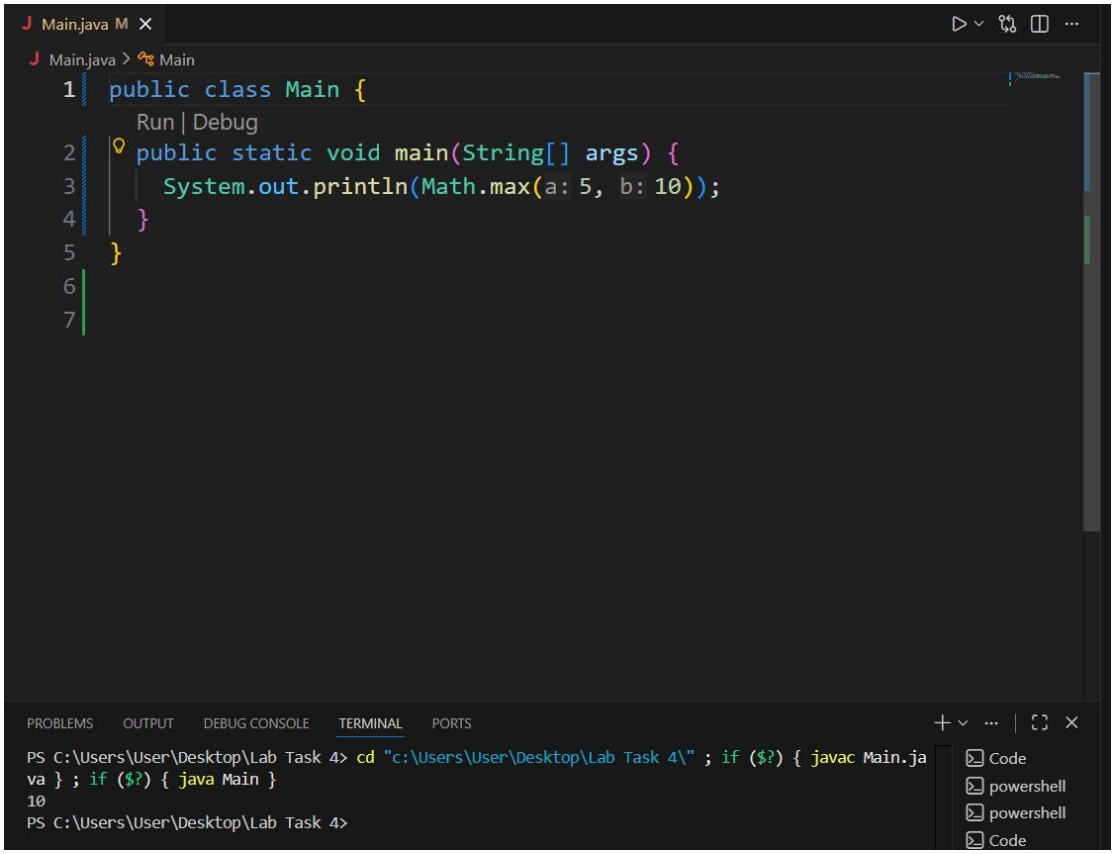


This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a String variable txt is created with extra spaces at the beginning and the end. The trim() method is used to remove all leading and trailing spaces from the string. The program then prints the string before and after using trim(), showing how the extra spaces are removed from the text.



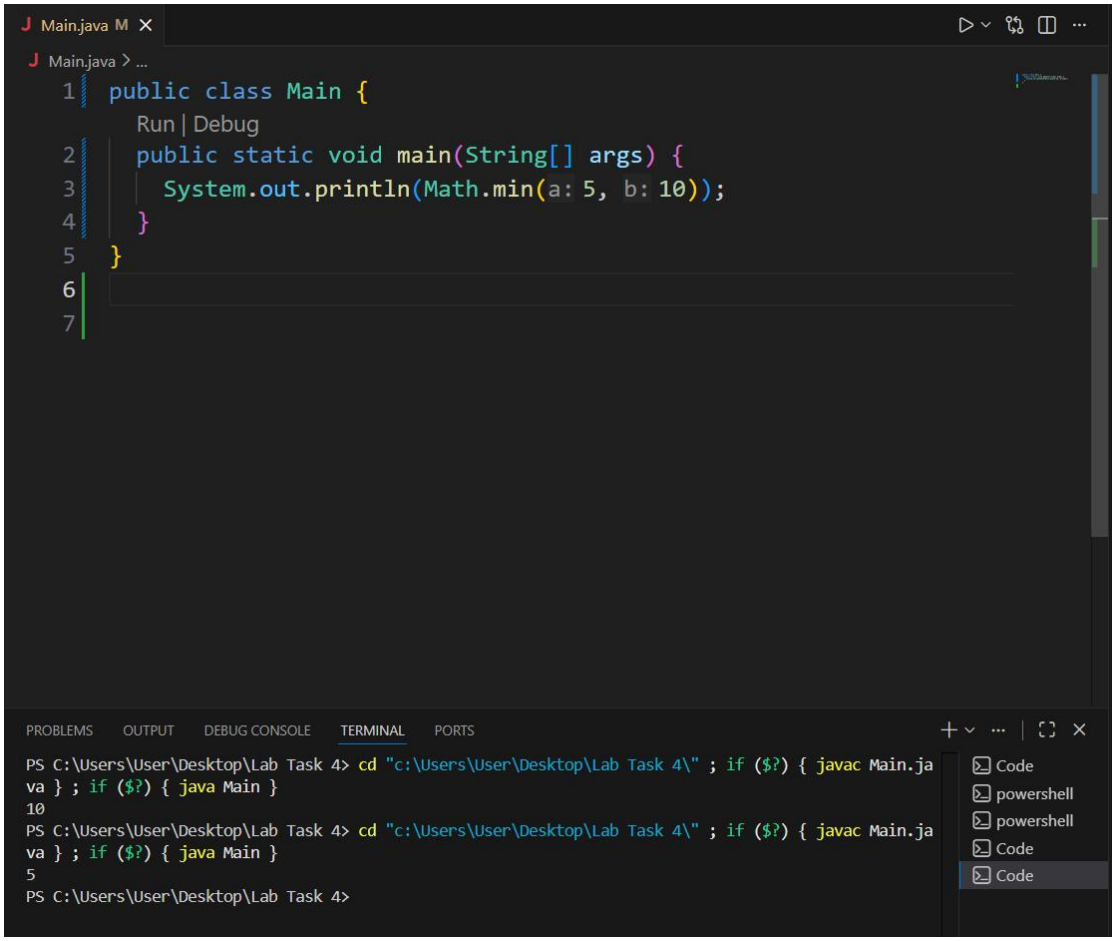
Math.max(X,Y)

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.max() method is used to find the larger of two numbers. Here, it compares 5 and 10, and returns the maximum value, which is 10. The result is then printed on the screen using System.out.println.



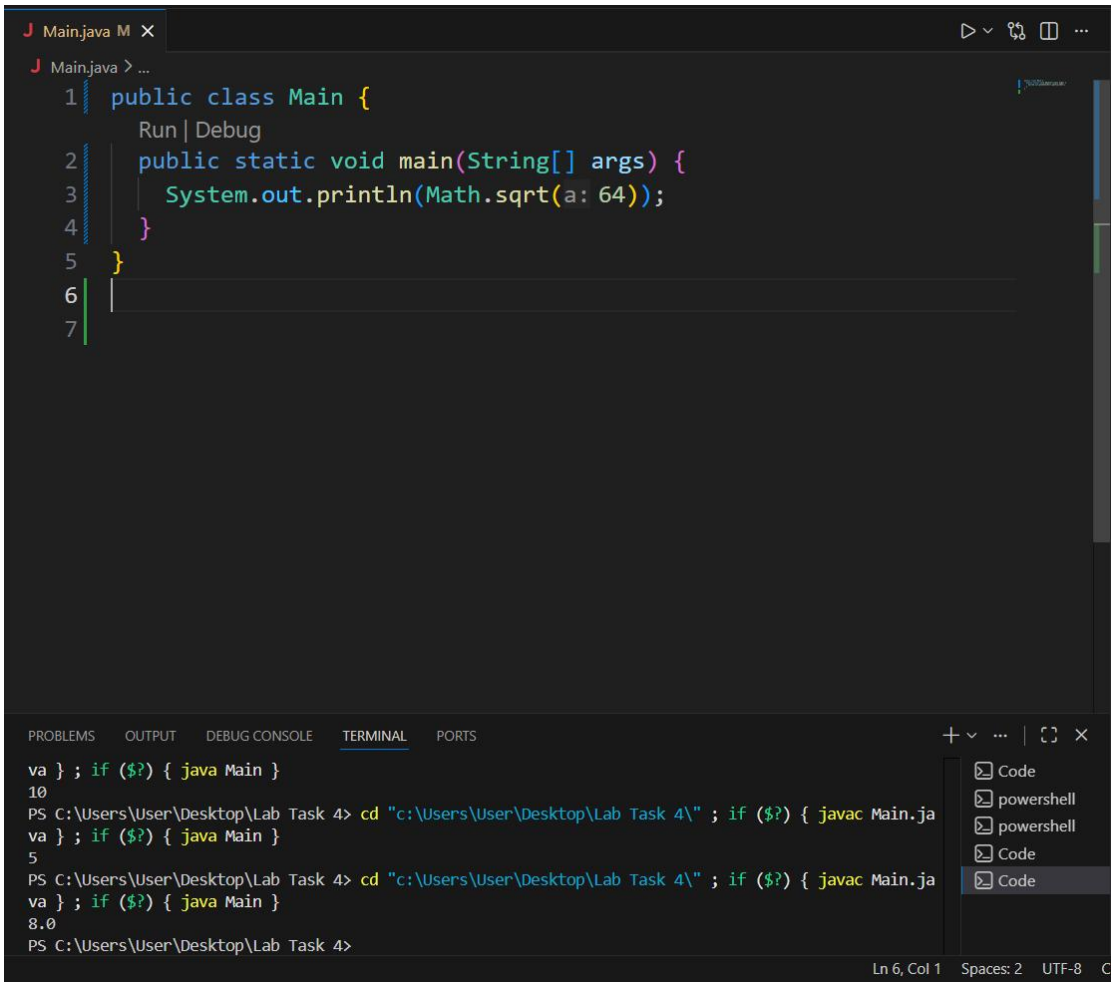
Math.min(X,Y)

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.min() method is used to find the smaller of two numbers. Here, it compares 5 and 10, and returns the minimum value, which is 5. The result is then printed on the screen using System.out.println.



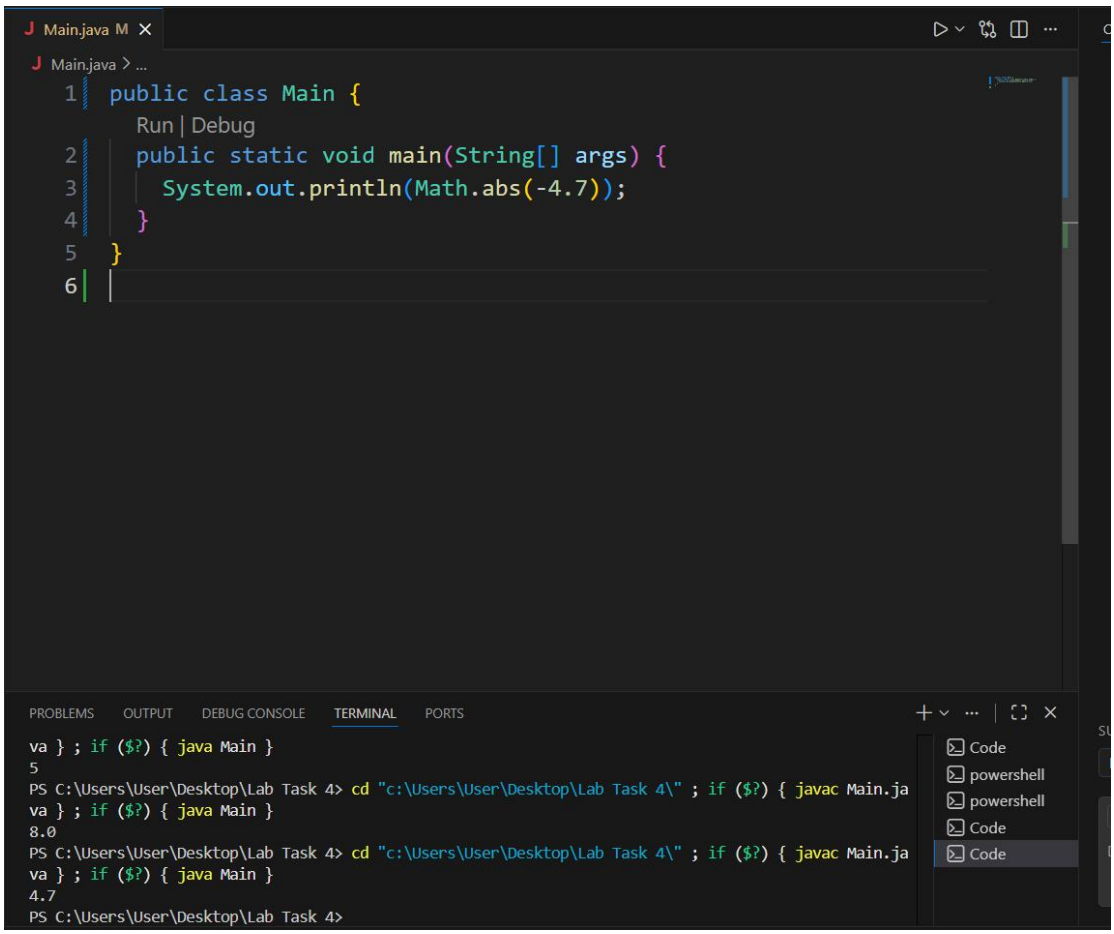
Math.SQRT(X)

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.sqrt() method is used to calculate the square root of a number. Here, it finds the square root of 64, which is 8. The result is then printed on the screen using System.out.println.



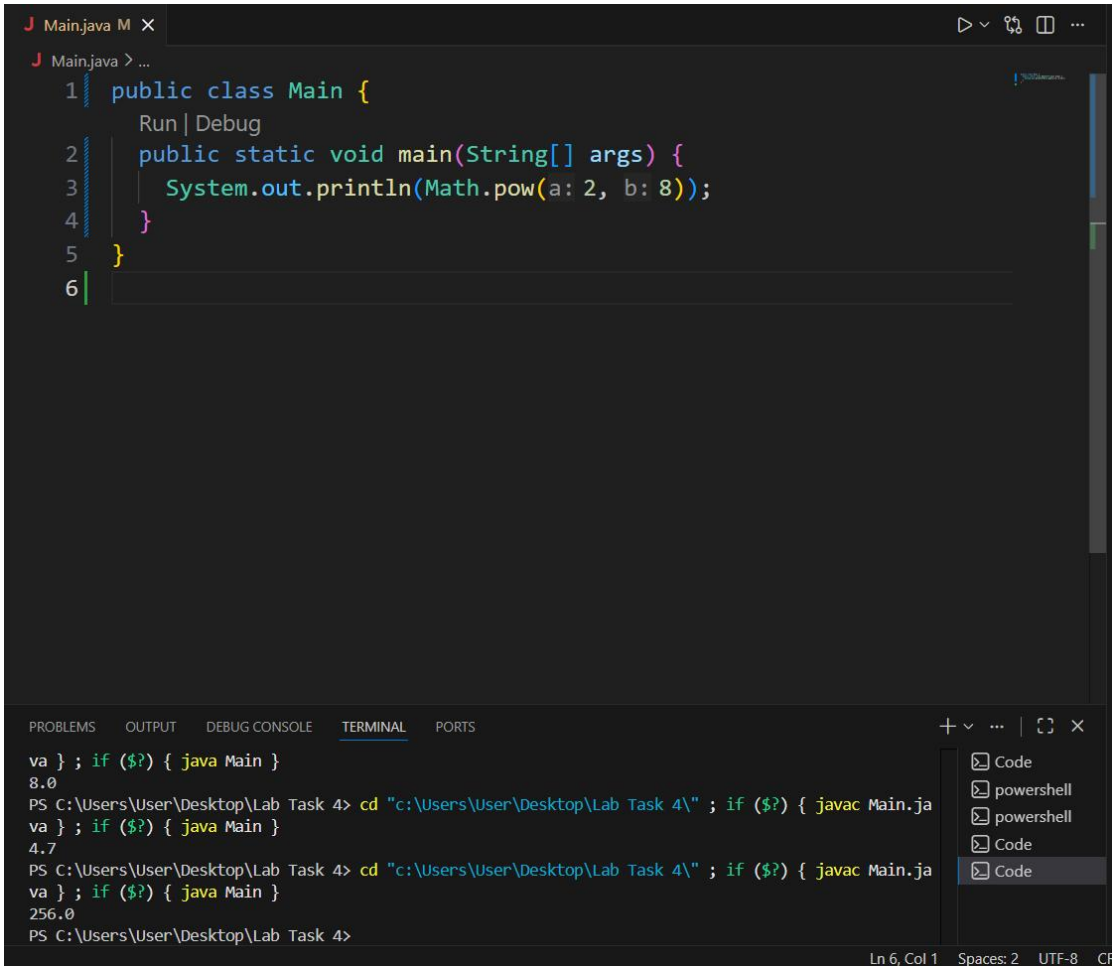
Math.abs(X)

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.abs() method is used to find the absolute value of a number, which means it converts negative numbers to positive. Here, it converts -4.7 to 4.7. The result is then printed on the screen using System.out.println.



Math.pow(X,Y)

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.pow() method is used to calculate the power of a number. Here, it raises 2 to the power of 8 (2^8), which equals 256. The result is then printed on the screen using System.out.println.



Rounding Methods

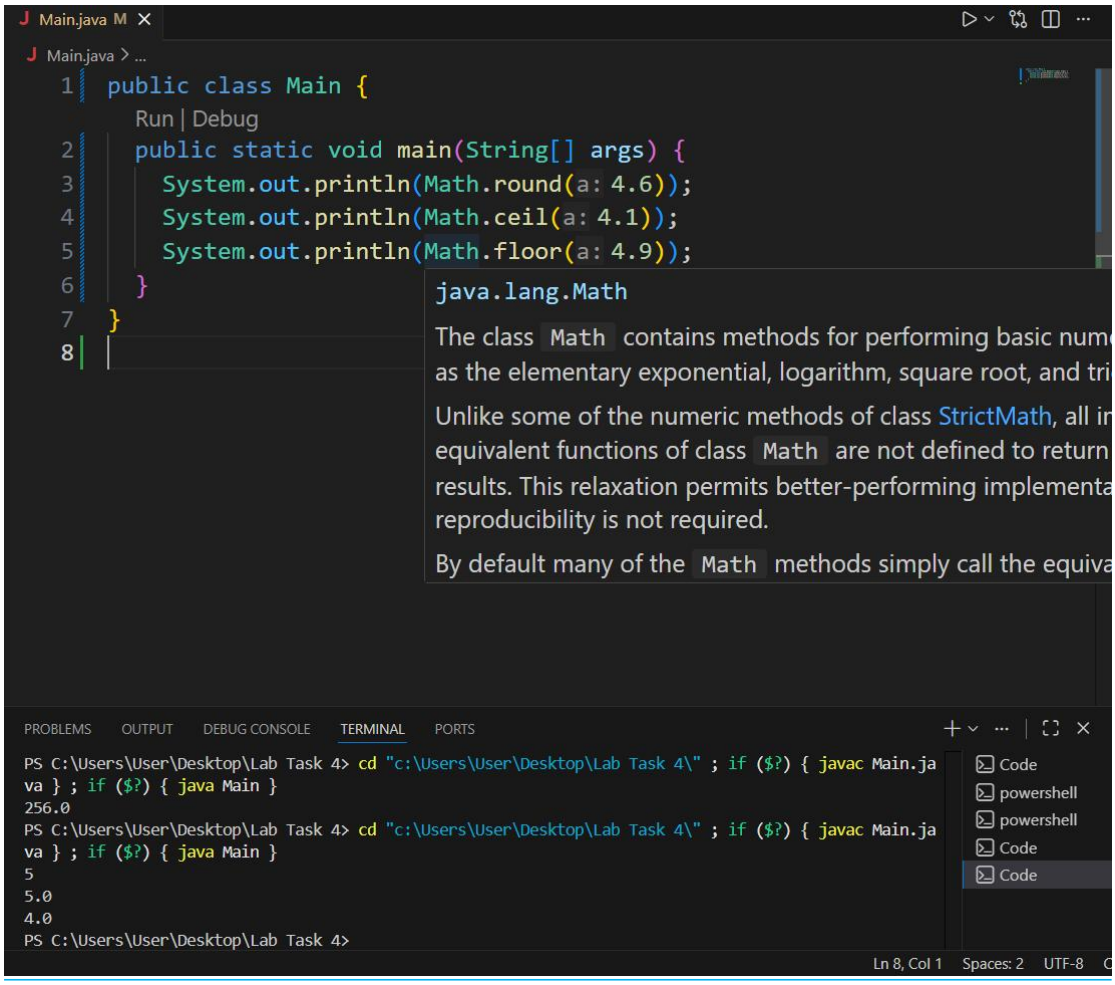
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, three `Math` methods are used to handle decimal numbers:

`Math.round(4.6)` rounds the number to the nearest whole number, giving `5`.

`Math.ceil(4.1)` rounds the number **up** to the next whole number, giving `5`.

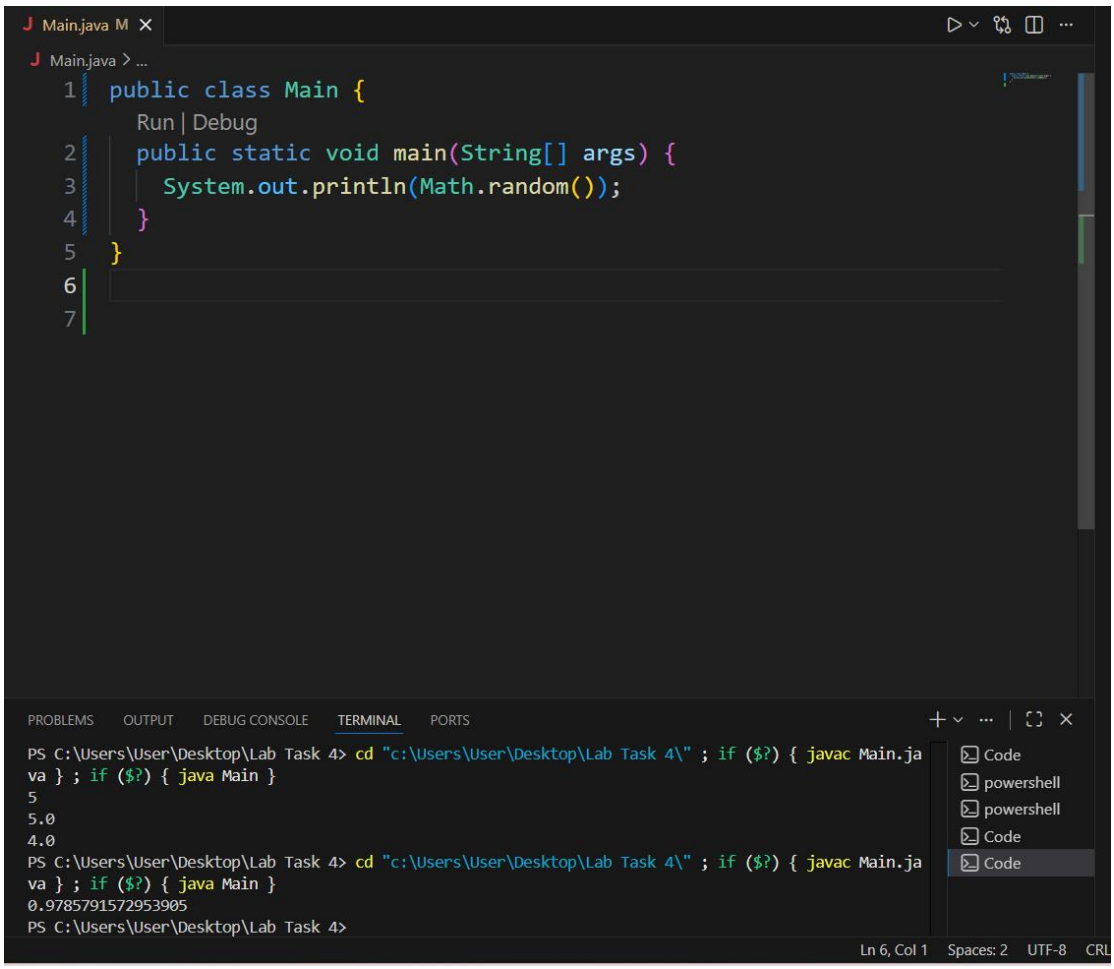
`Math.floor(4.9)` rounds the number **down** to the previous whole number, giving `4`.

All the results are printed on the screen using `System.out.println`.

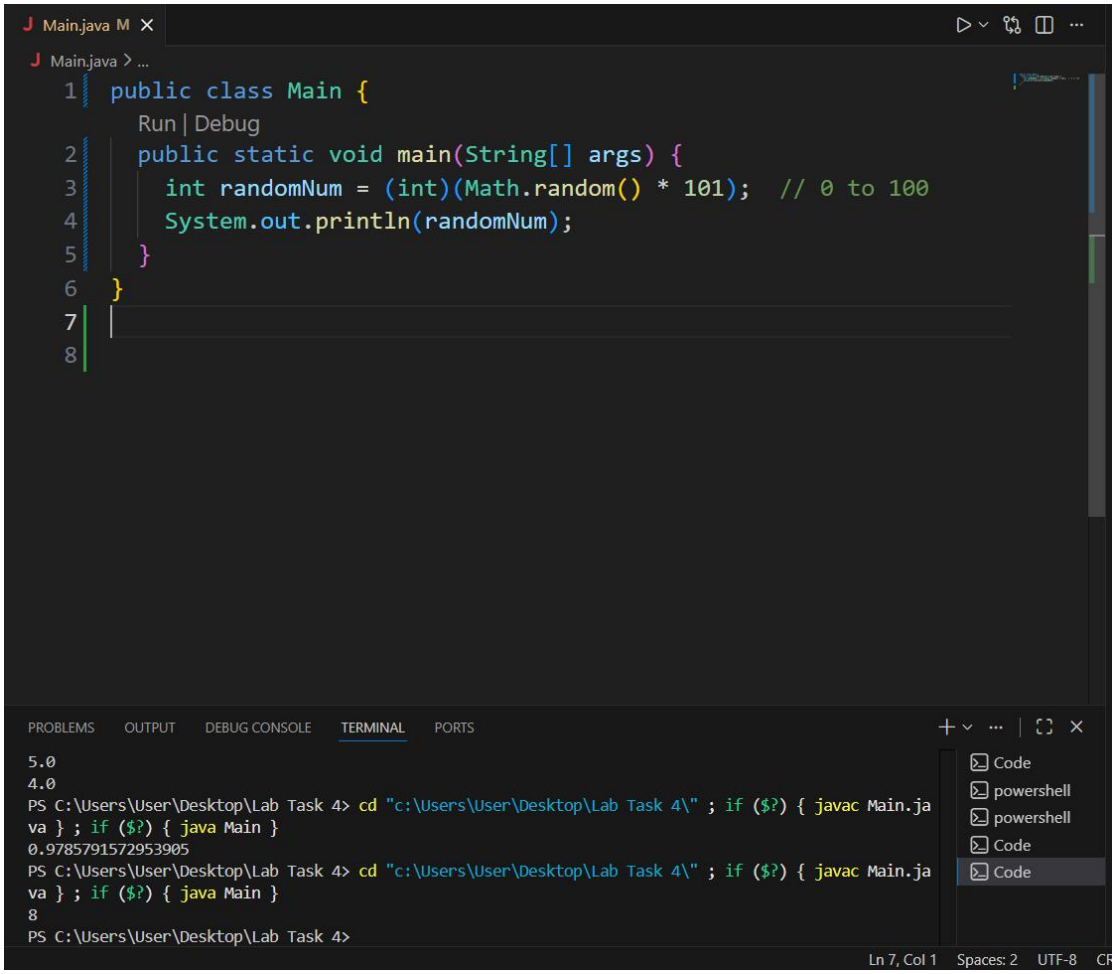


Random Numbers

This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the Math.random() method is used to generate a random decimal number between 0.0 (inclusive) and 1.0 (exclusive). The generated random number is then printed on the screen using System.out.println.

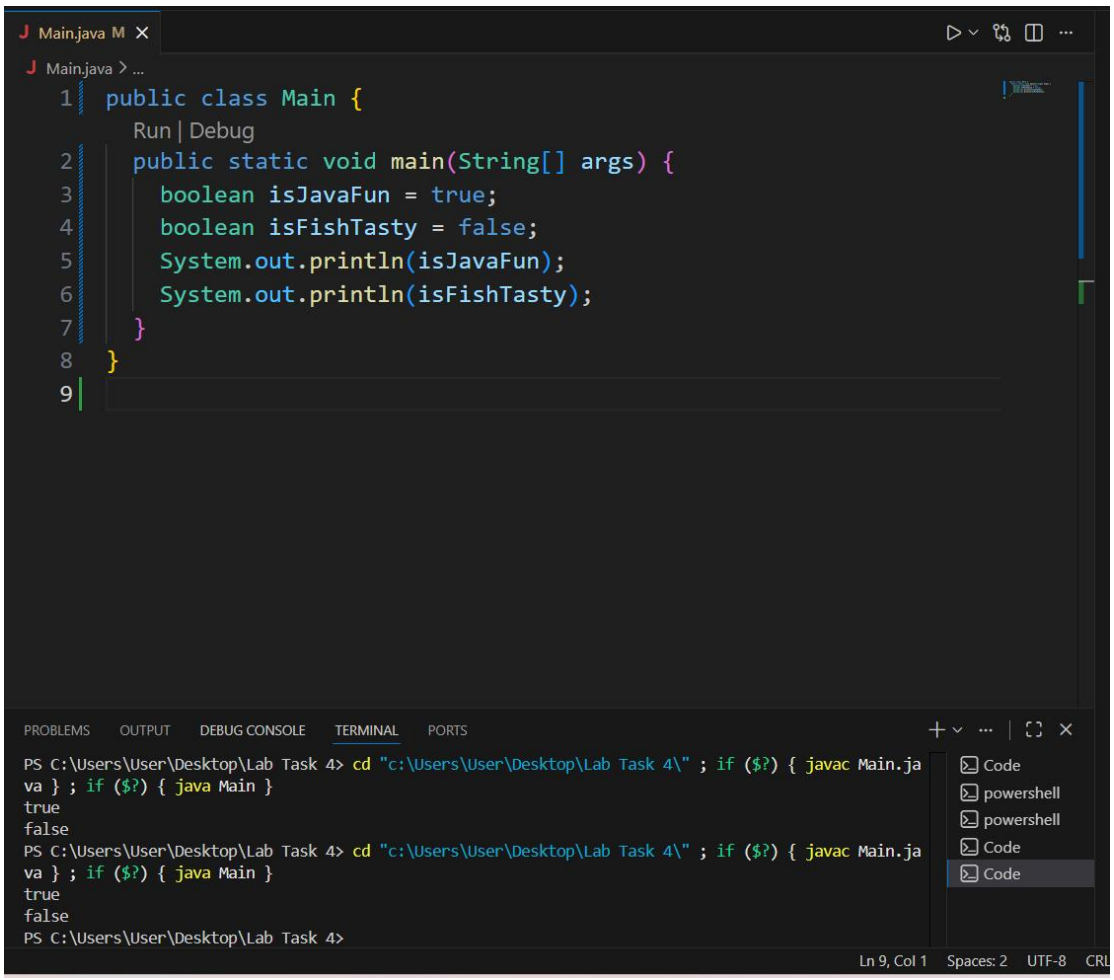


This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a random number between 0 and 100 is generated using Math.random(). Multiplying Math.random() by 101 gives a decimal number from 0.0 up to 100.999…, and casting it to (int) converts it to an integer between 0 and 100. The generated random number is then printed on the screen using System.out.println.

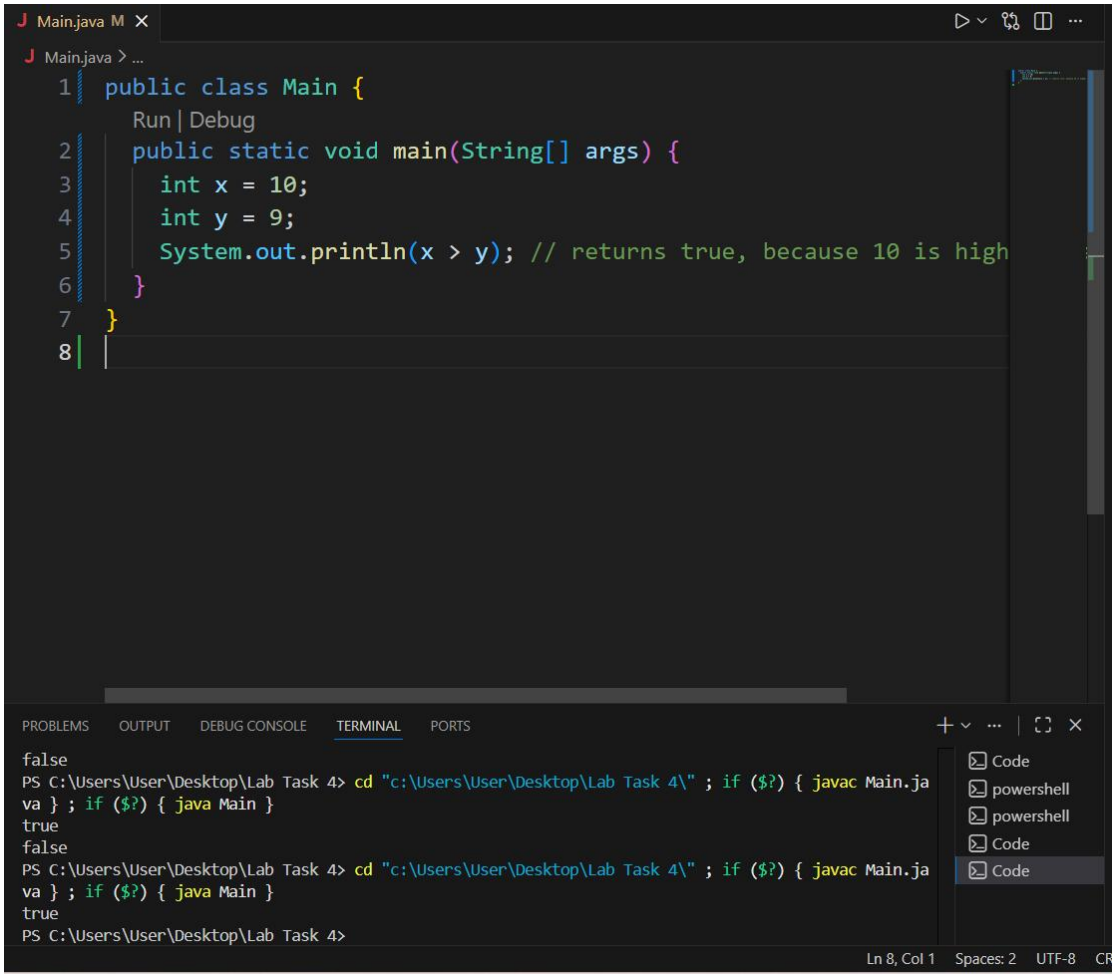


Java Booleans

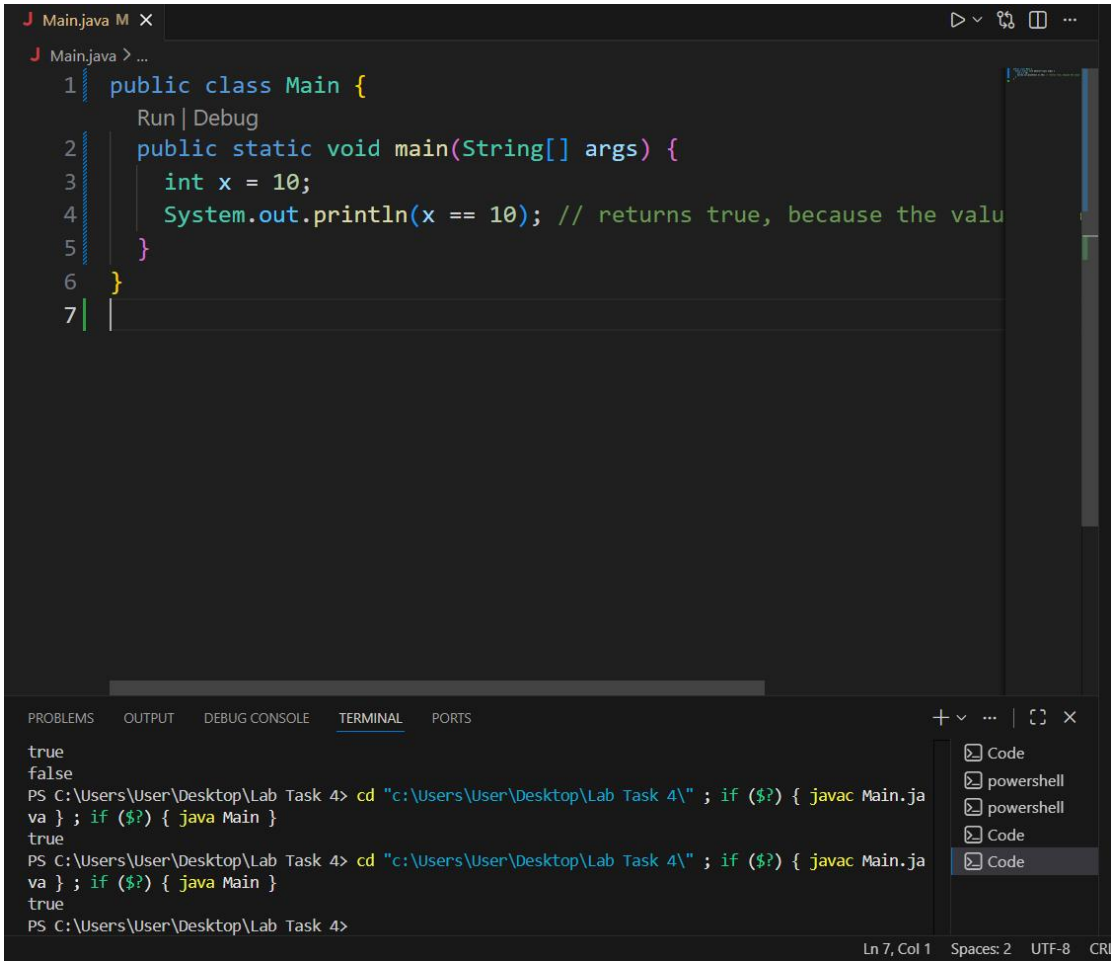
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two boolean variables are created: isJavaFun with the value true, and isFishTasty with the value false. Boolean variables can only hold true or false. The program then prints the values of both variables on the screen using System.out.println.



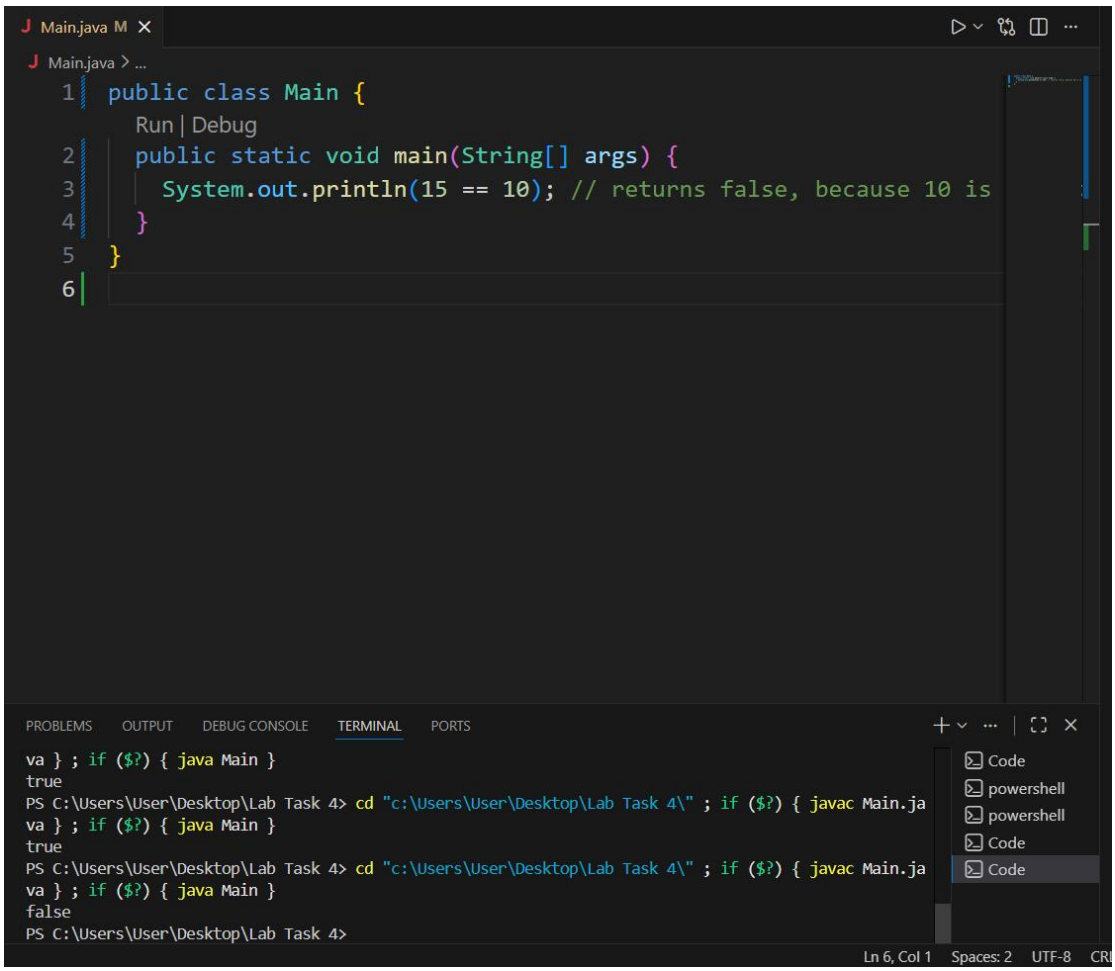
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two integer variables x and y are created with values 10 and 9. The expression x > y checks if x is greater than y. Since 10 is greater than 9, the expression is true. The result is then printed on the screen using System.out.println.



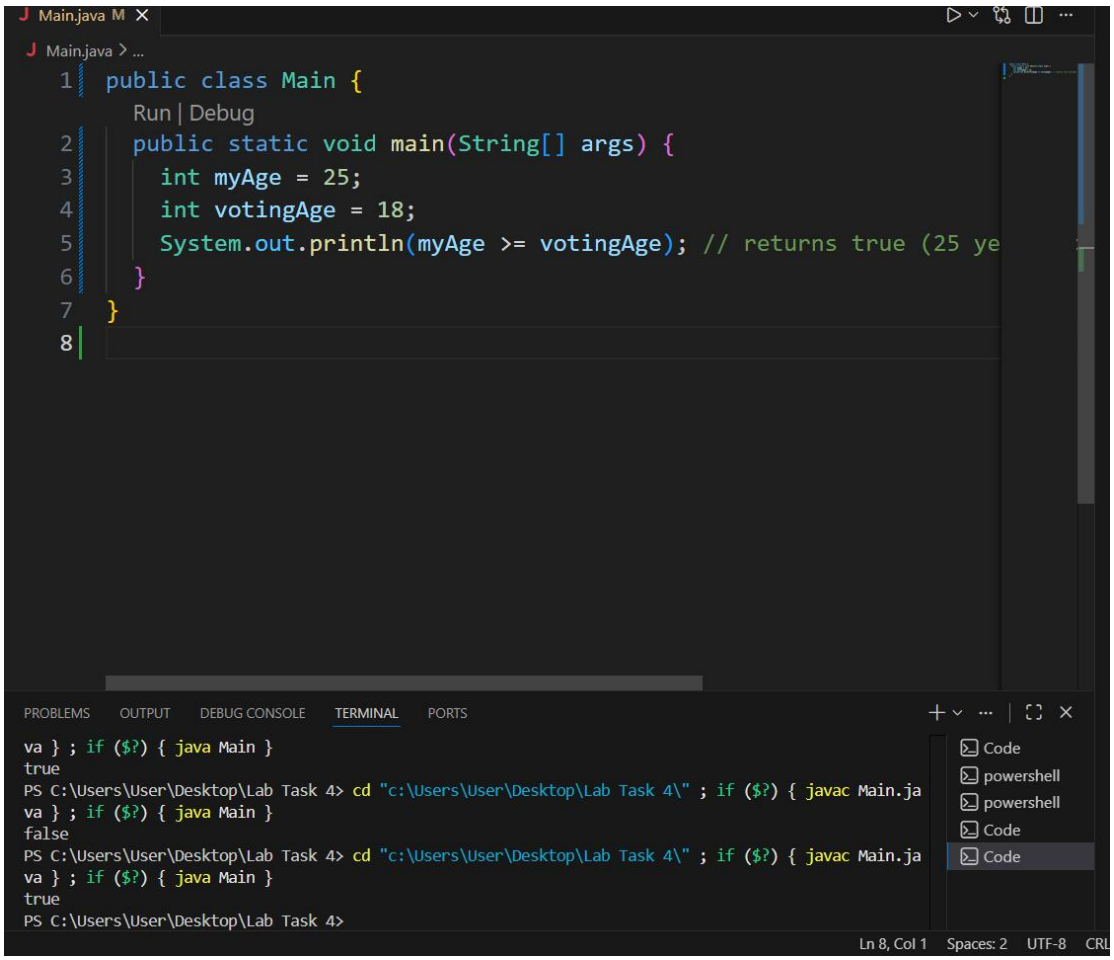
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, an integer variable x is created and assigned the value 10. The expression x == 10 checks if x is equal to 10. Since x is indeed 10, the expression is true. The result is then printed on the screen using System.out.println.



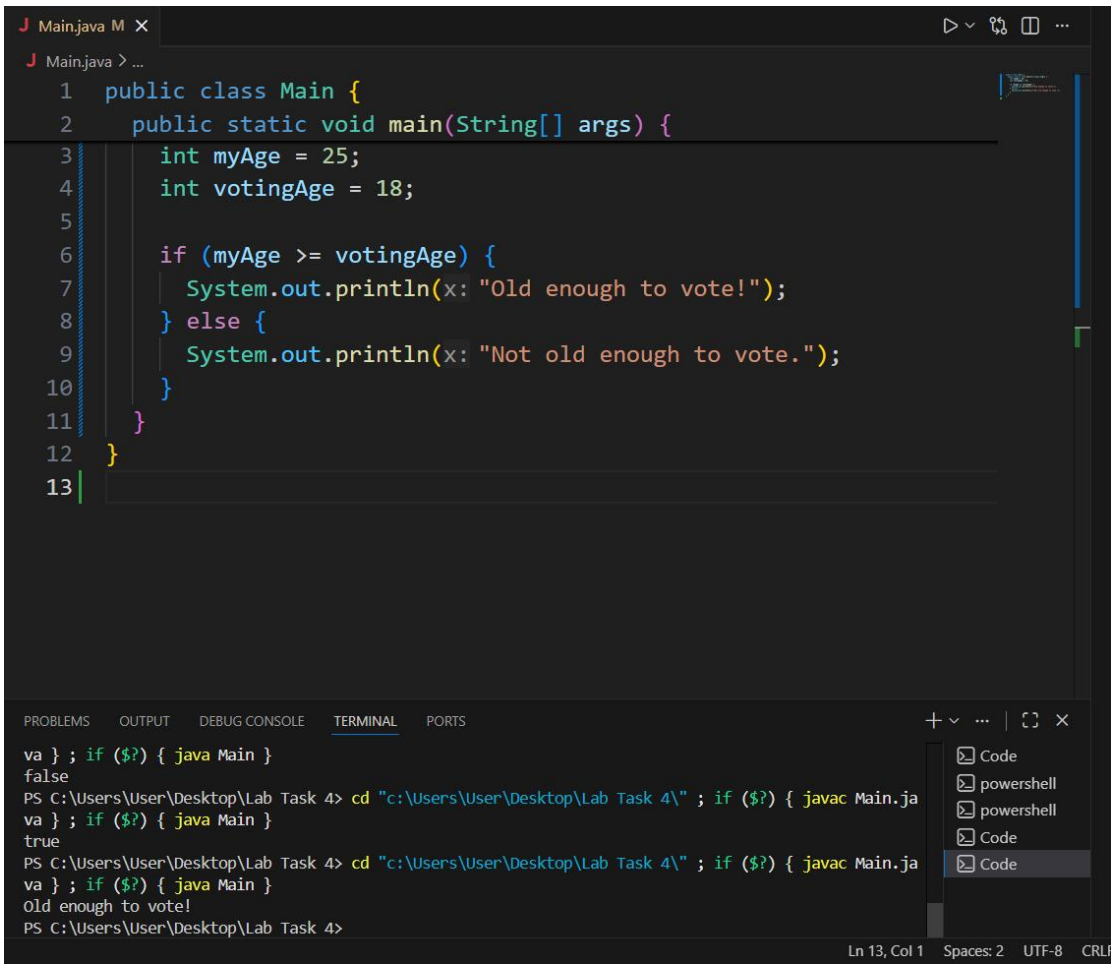
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, the expression 15 == 10 checks if 15 is equal to 10. Since 15 is not equal to 10, the expression is false. The result is then printed on the screen using System.out.println.



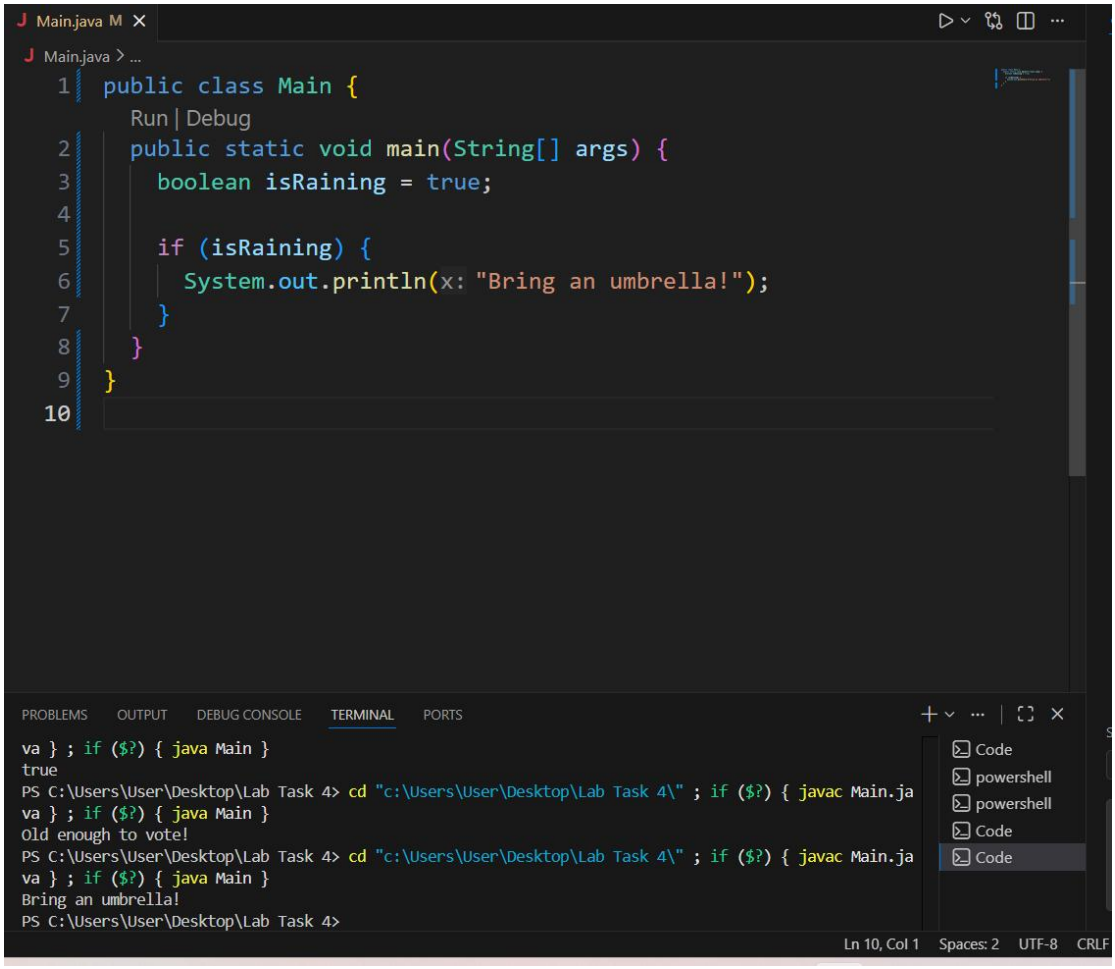
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two integer variables are created: myAge with the value 25 and votingAge with the value 18. The expression myAge >= votingAge checks if myAge is greater than or equal to votingAge. Since 25 is greater than 18, the expression is true. The result is then printed on the screen using System.out.println.



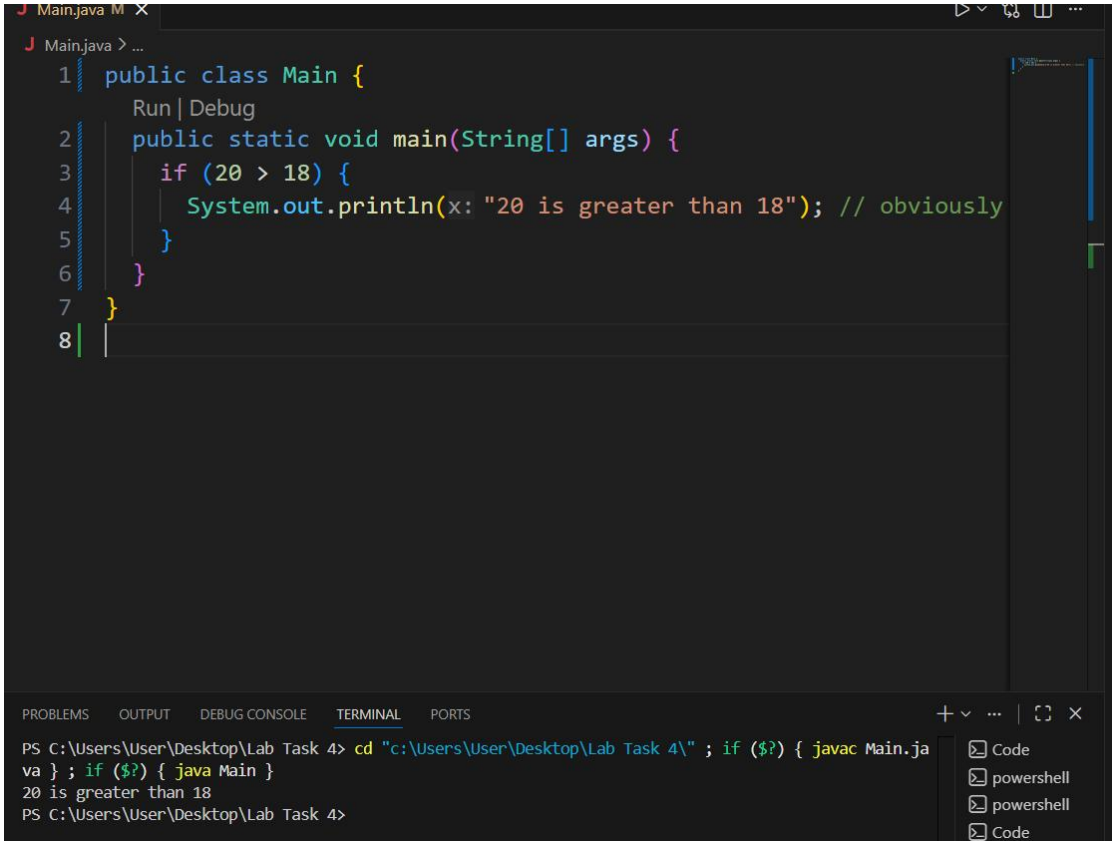
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two integer variables are created: myAge with the value 25 and votingAge with the value 18. The program uses an if-else statement to check if myAge is greater than or equal to votingAge. If the condition is true, it prints "Old enough to vote!"; otherwise, it prints "Not old enough to vote.". Since 25 is greater than 18, the program prints "Old enough to vote!".



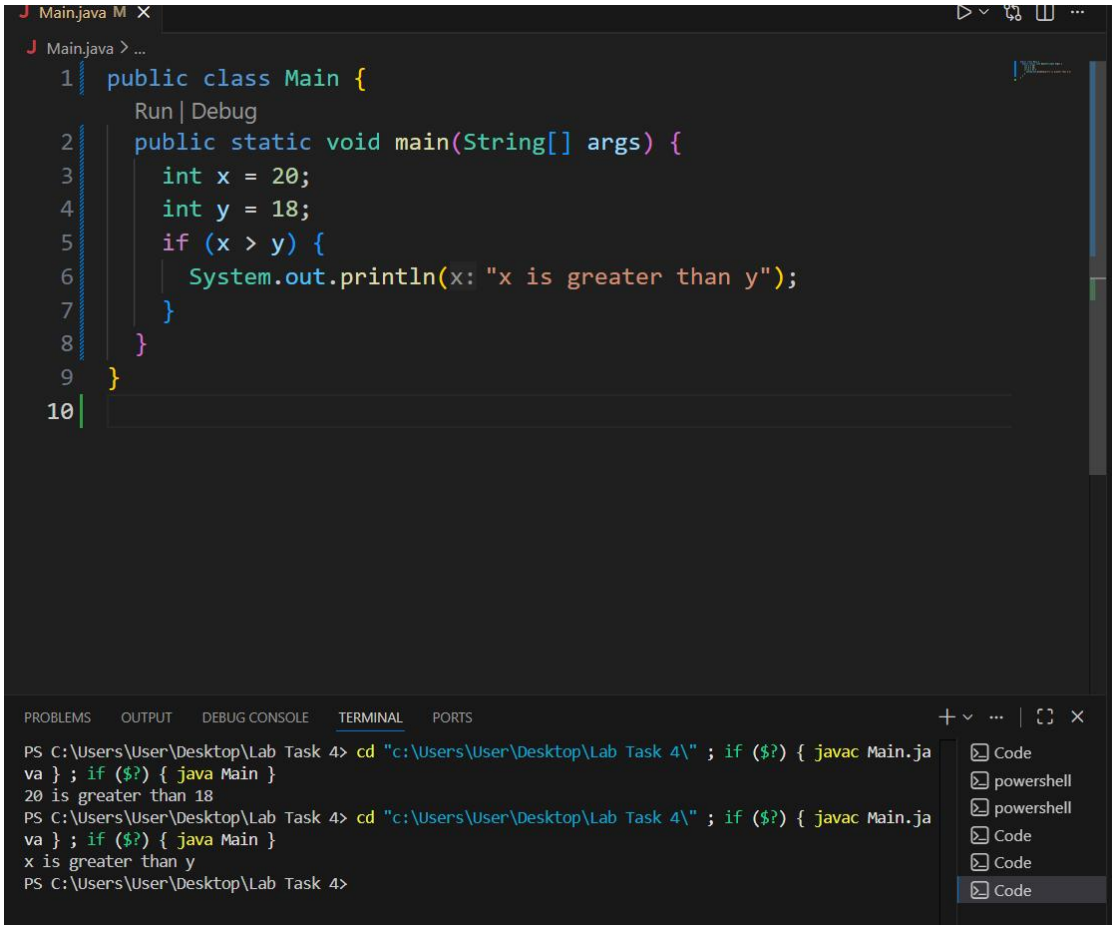
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a boolean variable isRaining is created and set to true. The program uses an if statement to check the value of isRaining. If it is true, it prints "Bring an umbrella!" on the screen. Since isRaining is true, the message is displayed.



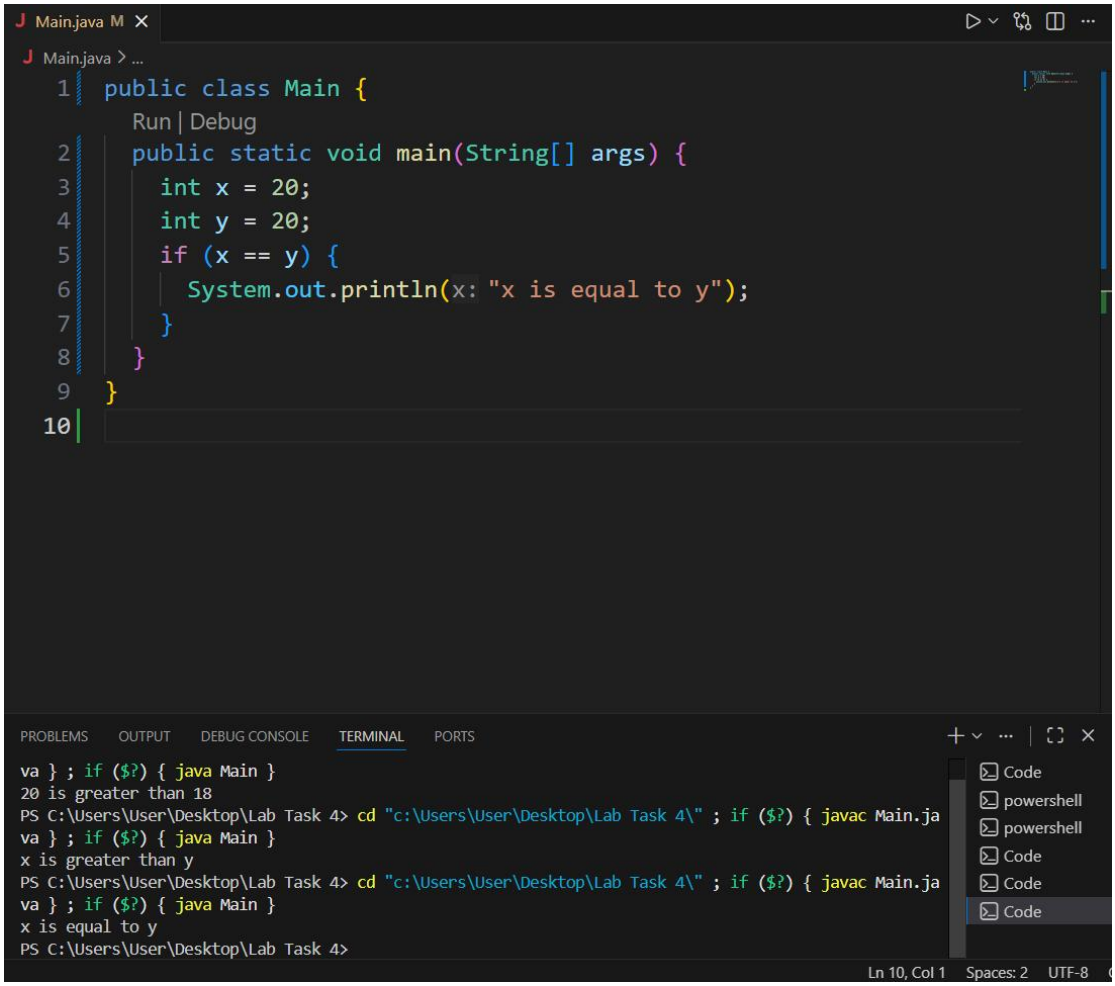
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an `if` statement is used to check the condition `20 > 18`. Since `20` is greater than `18`, the condition is true, and the program prints the message `"20 is greater than 18"` on the screen.



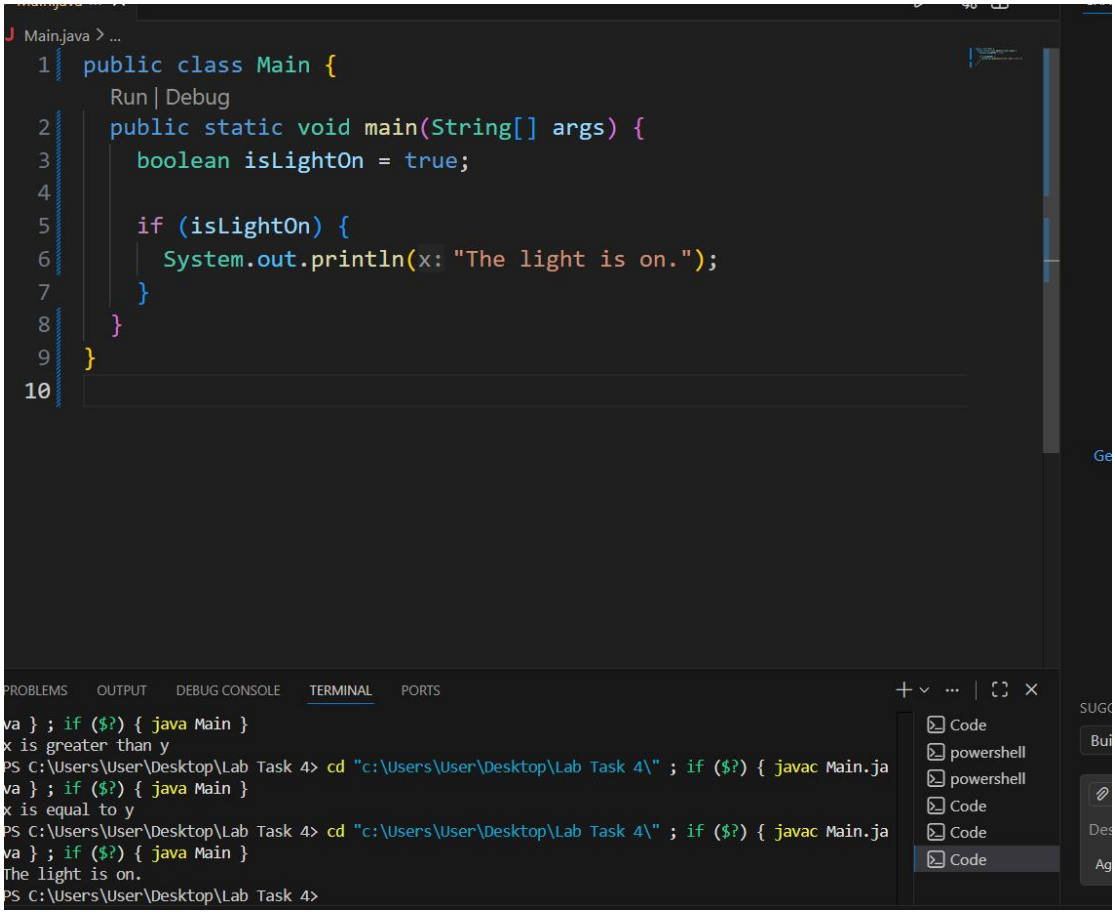
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two integer variables x and y are created with values 20 and 18. The if statement checks whether x is greater than y. Since 20 is greater than 18, the condition is true and the program prints the message "x is greater than y" on the screen.



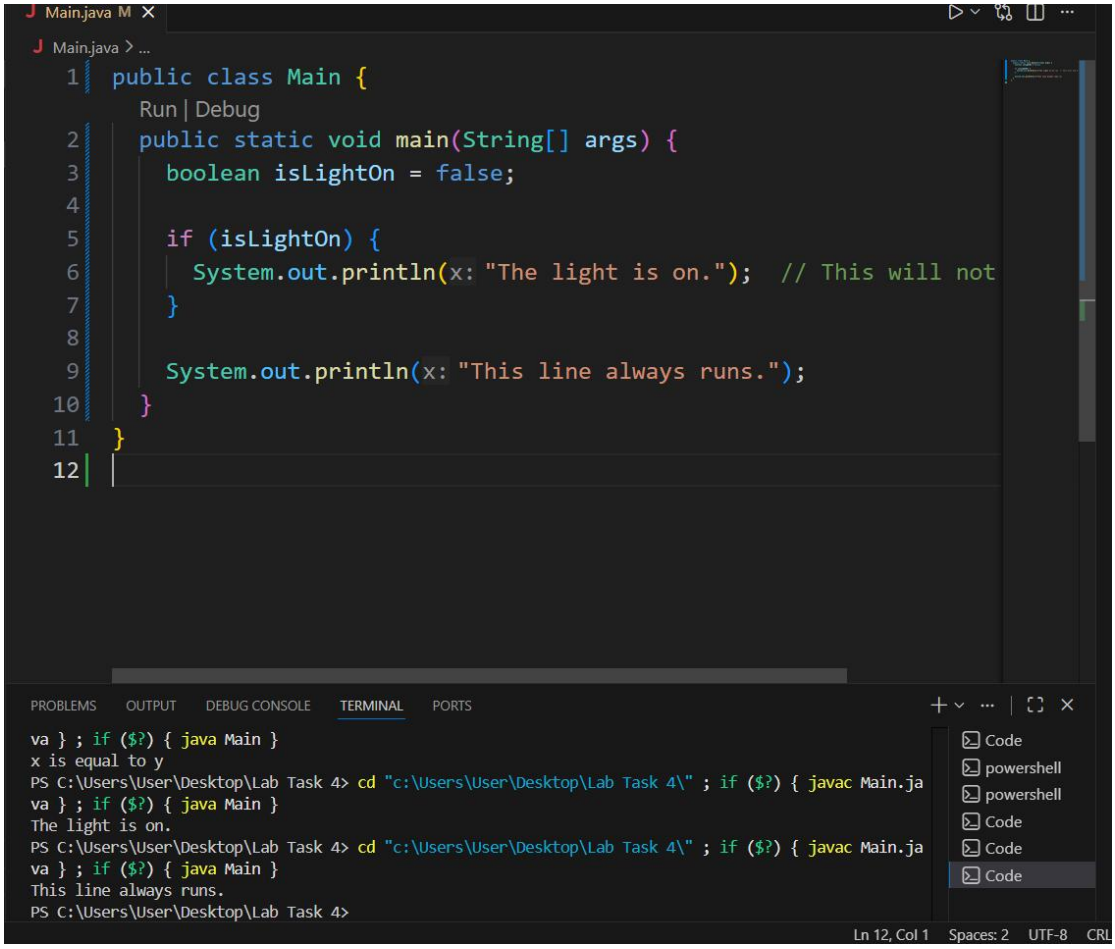
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, two integer variables x and y are created with the same value 20. The if statement checks whether x is equal to y using the == operator. Since both values are equal, the condition is true and the program prints the message "x is equal to y" on the screen.



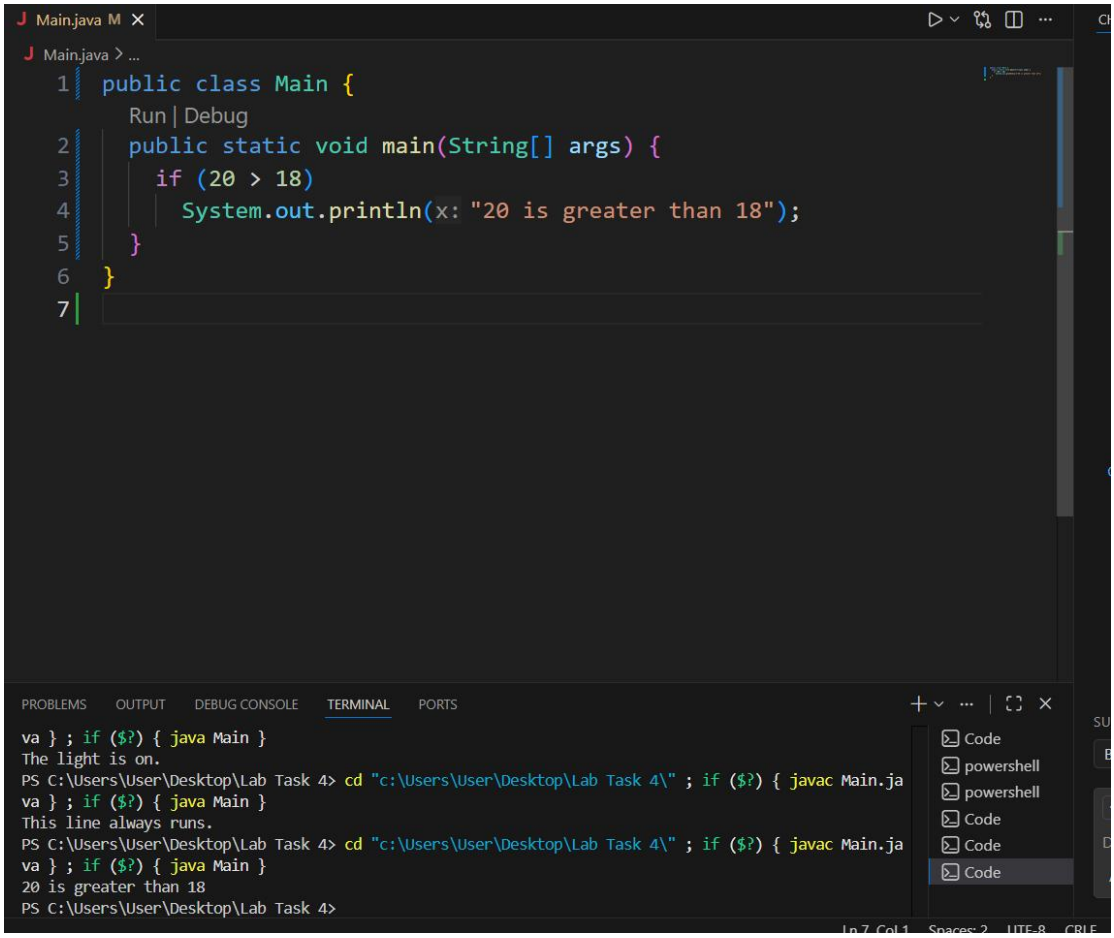
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a boolean variable `isLightOn` is created and set to `true`. The `if` statement checks the value of `isLightOn`. Since it is `true`, the condition is satisfied and the program prints the message `"The light is on."` on the screen.



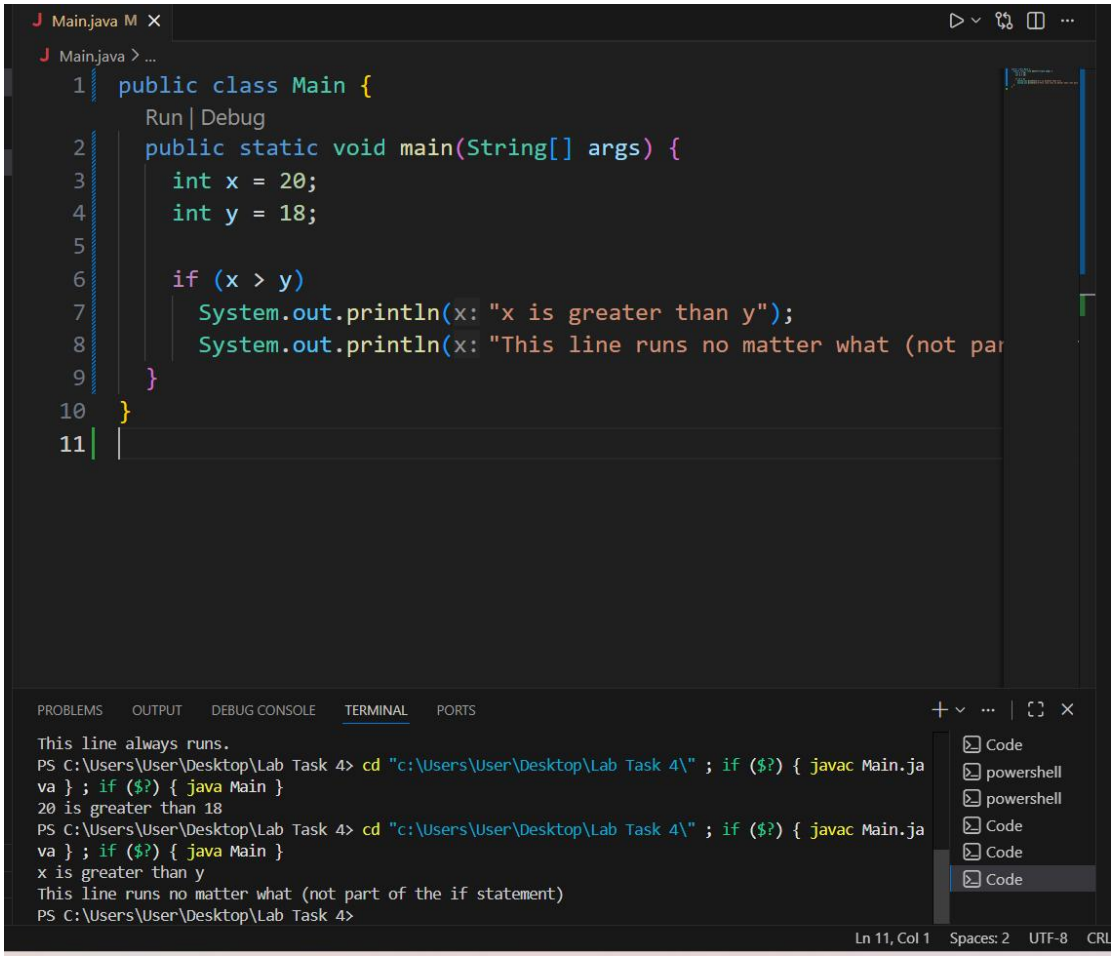
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a boolean variable isLightOn is created and set to false. The if statement checks whether isLightOn is true. Since it is false, the message "The light is on." is not printed. However, the last System.out.println statement is outside the if block, so it always runs and prints "This line always runs." on the screen.



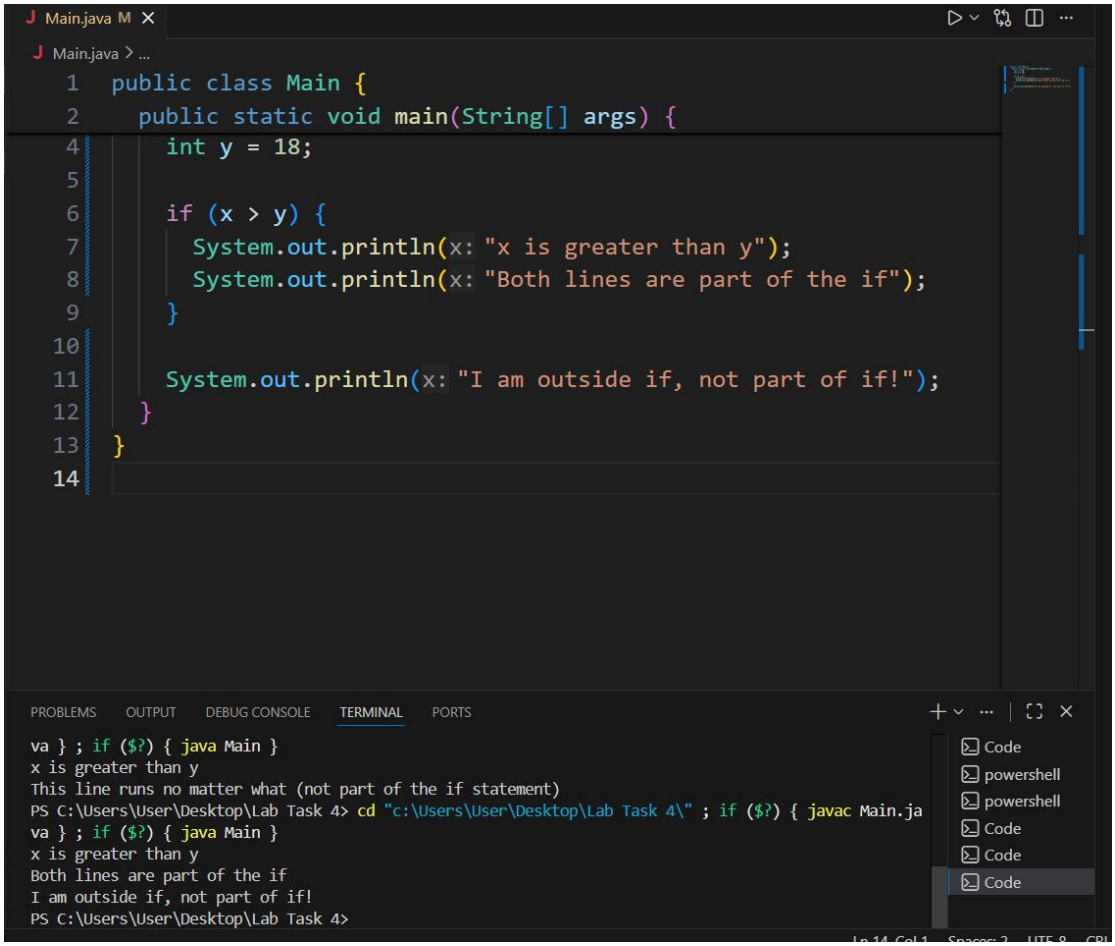
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, an if statement is used without curly braces because it contains only one statement. The condition 20 > 18 is checked, and since it is true, the program prints the message "20 is greater than 18" on the screen.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two integer variables `x` and `y` are created with values `20` and `18`. The `if` statement checks whether `x` is greater than `y`. Since there are no curly braces, only the first `System.out.println` line is controlled by the `if` statement. The second `System.out.println` line is not part of the `if` statement, so it runs every time, regardless of whether the condition is true or false.

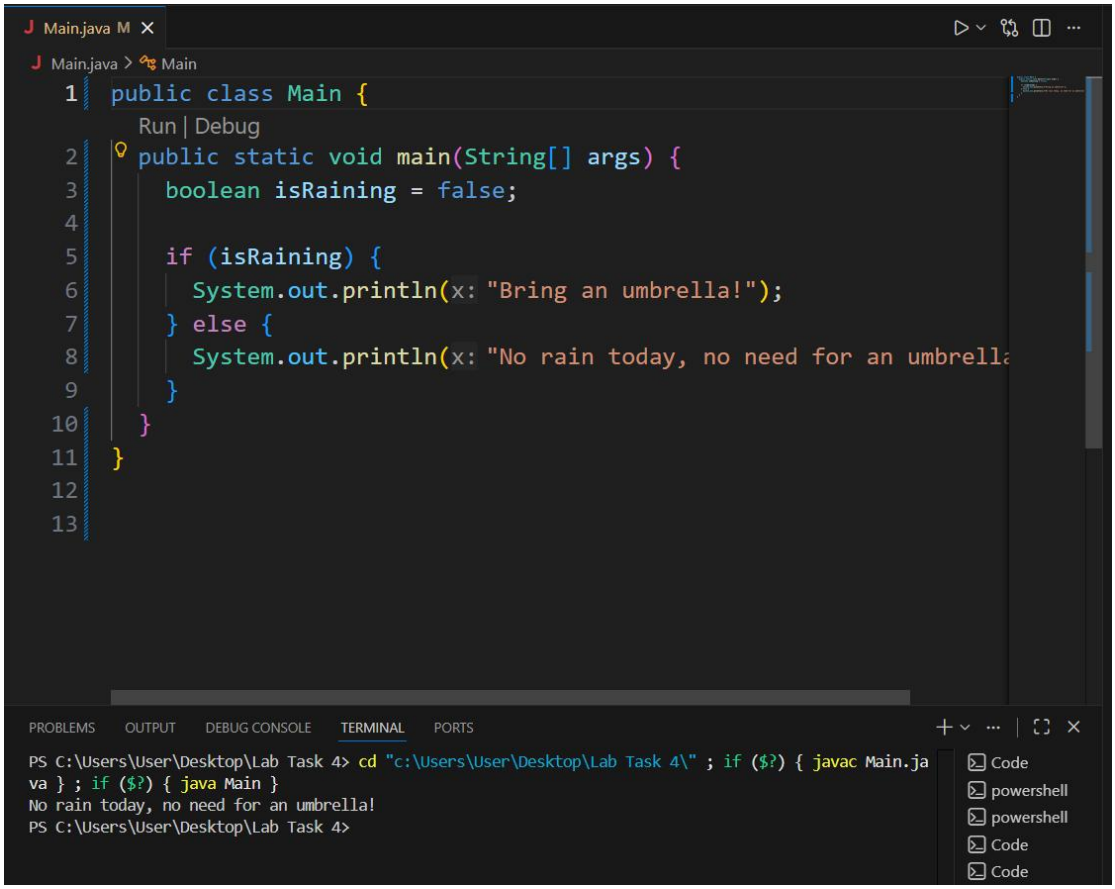


This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two integer variables `x` and `y` are created with values `20` and `18`. The `if` statement checks whether `x` is greater than `y`. Because curly braces are used, both `System.out.println` statements inside the braces are part of the `if` block and run only when the condition is true. The last `System.out.println` statement is outside the `if` block, so it always runs, regardless of the condition.

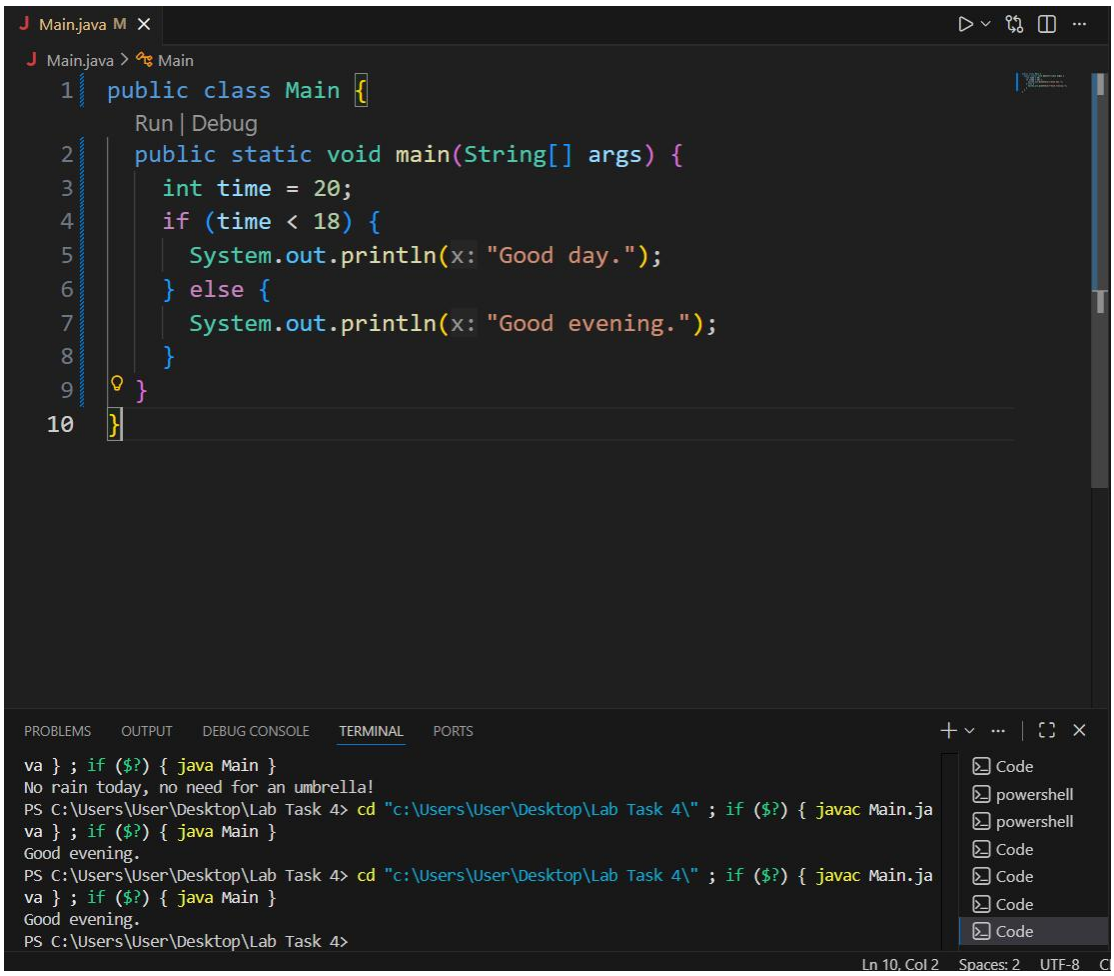


Java Else Statements

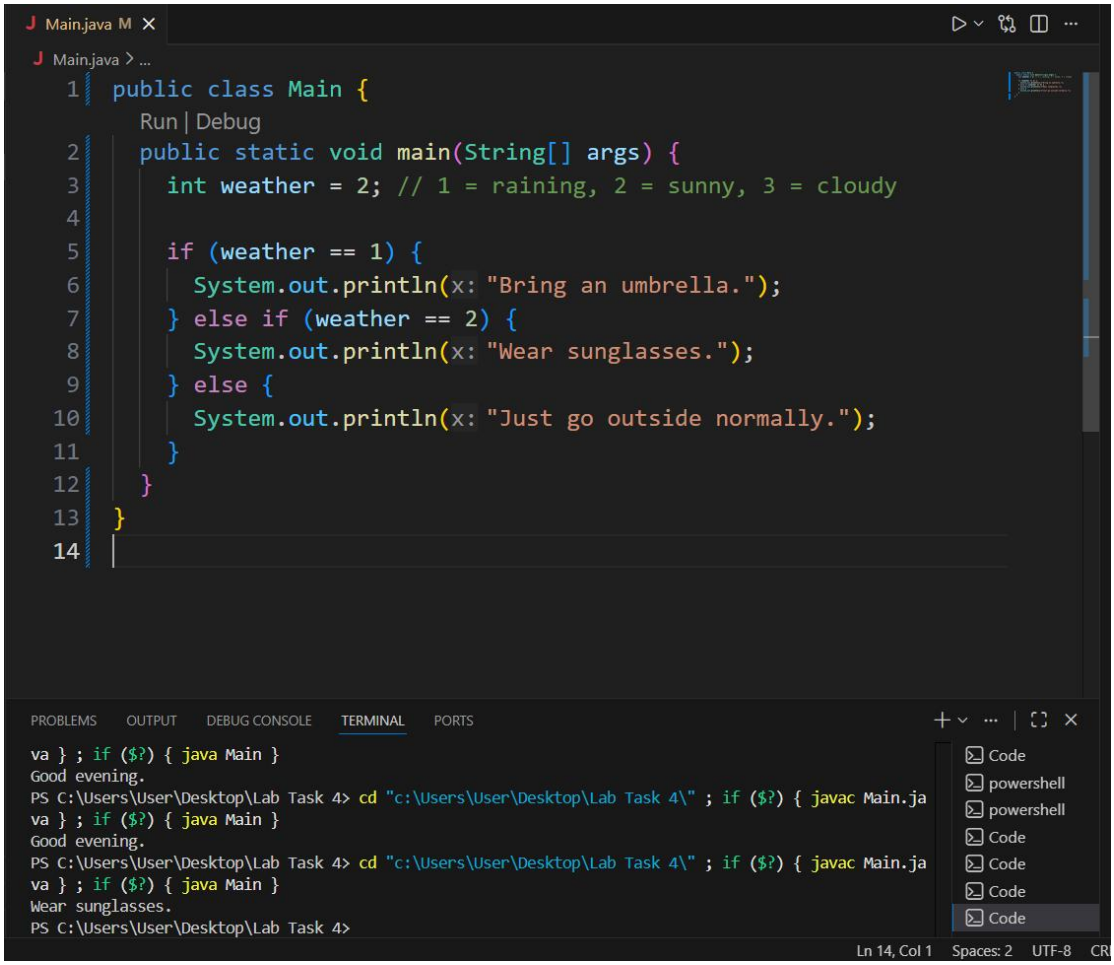
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a boolean variable `isRaining` is created and set to `false`. The program uses an `if-else` statement to check whether it is raining. If `isRaining` is `true`, it prints `"Bring an umbrella!"`. Otherwise, it prints `"No rain today, no need for an umbrella!"`. Since `isRaining` is `false`, the `else` part runs and the second message is printed.



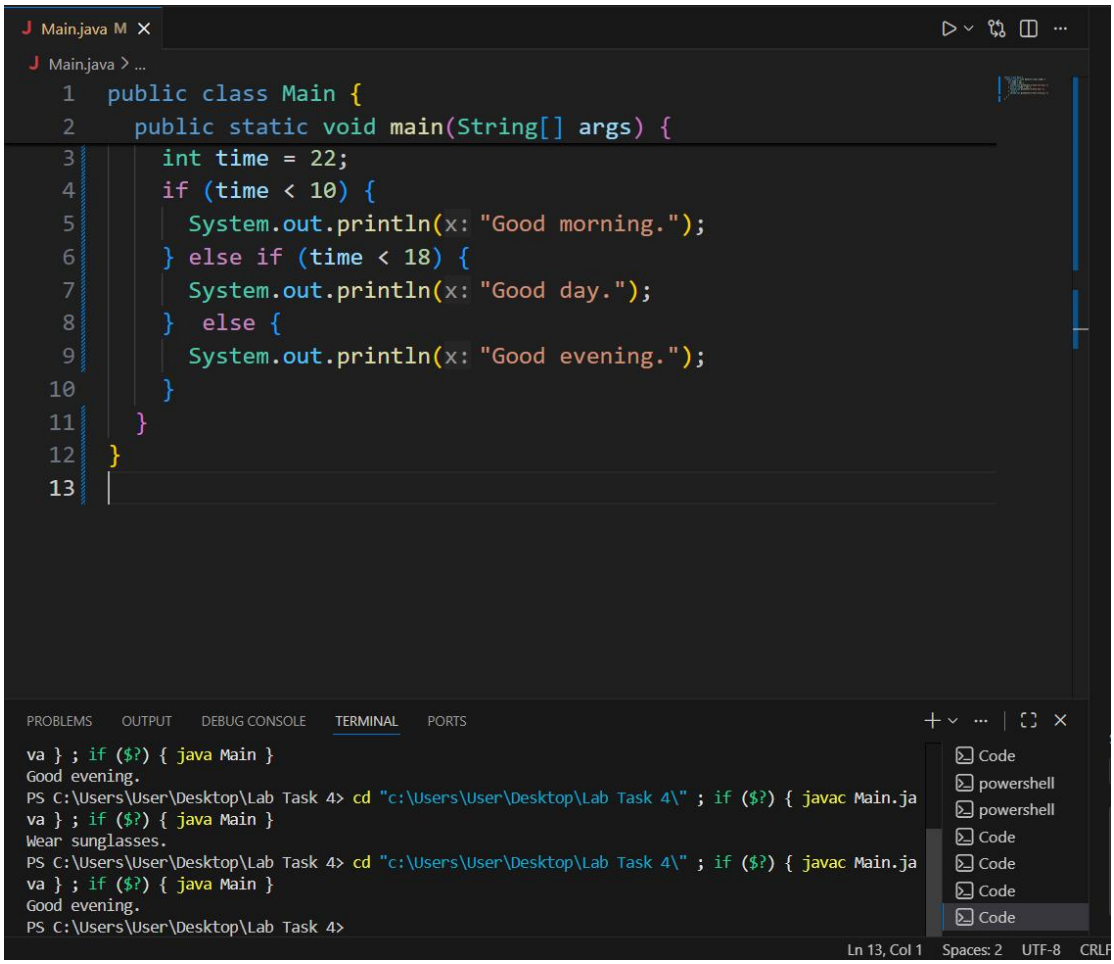
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, an integer variable time is created with the value 20. The if-else statement checks whether time is less than 18. If the condition is true, the program prints "Good day."; otherwise, it prints "Good evening.". Since time is 20, which is greater than 18, the else block runs and "Good evening." is printed.



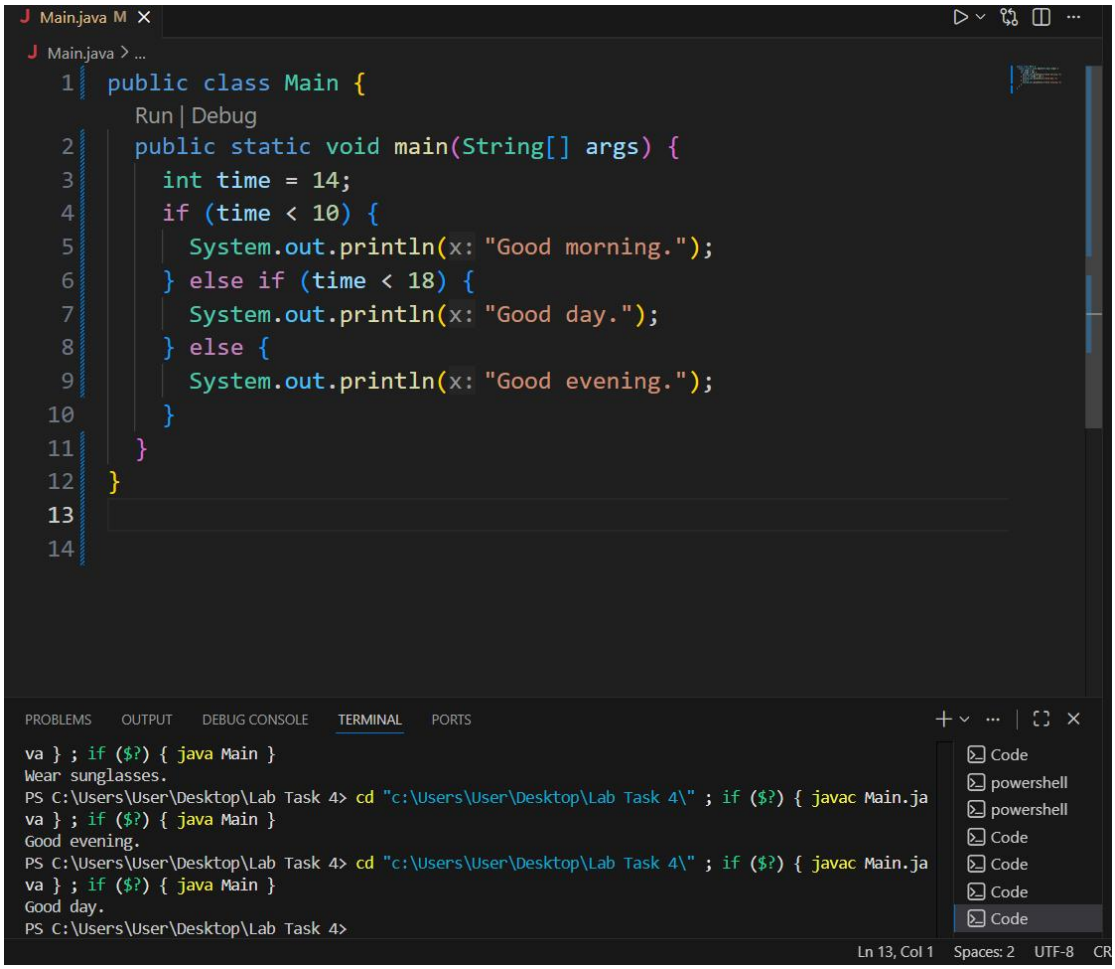
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, an integer variable weather is created to represent different weather conditions, where 1 means raining, 2 means sunny, and 3 means cloudy. The program uses an if-else if-else statement to check the value of weather. If weather is 1, it prints "Bring an umbrella.". If weather is 2, it prints "Wear sunglasses.". For any other value, it prints "Just go outside normally.". Since weather is 2, the program prints "Wear sunglasses.".



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `time` is created with the value `22`. The program uses an `if-else if-else` statement to check different time ranges. If `time` is less than `10`, it prints `"Good morning."`. If `time` is less than `18`, it prints `"Good day."`. Otherwise, it prints `"Good evening."`. Since `time` is `22`, the last condition is executed and `"Good evening."` is printed.

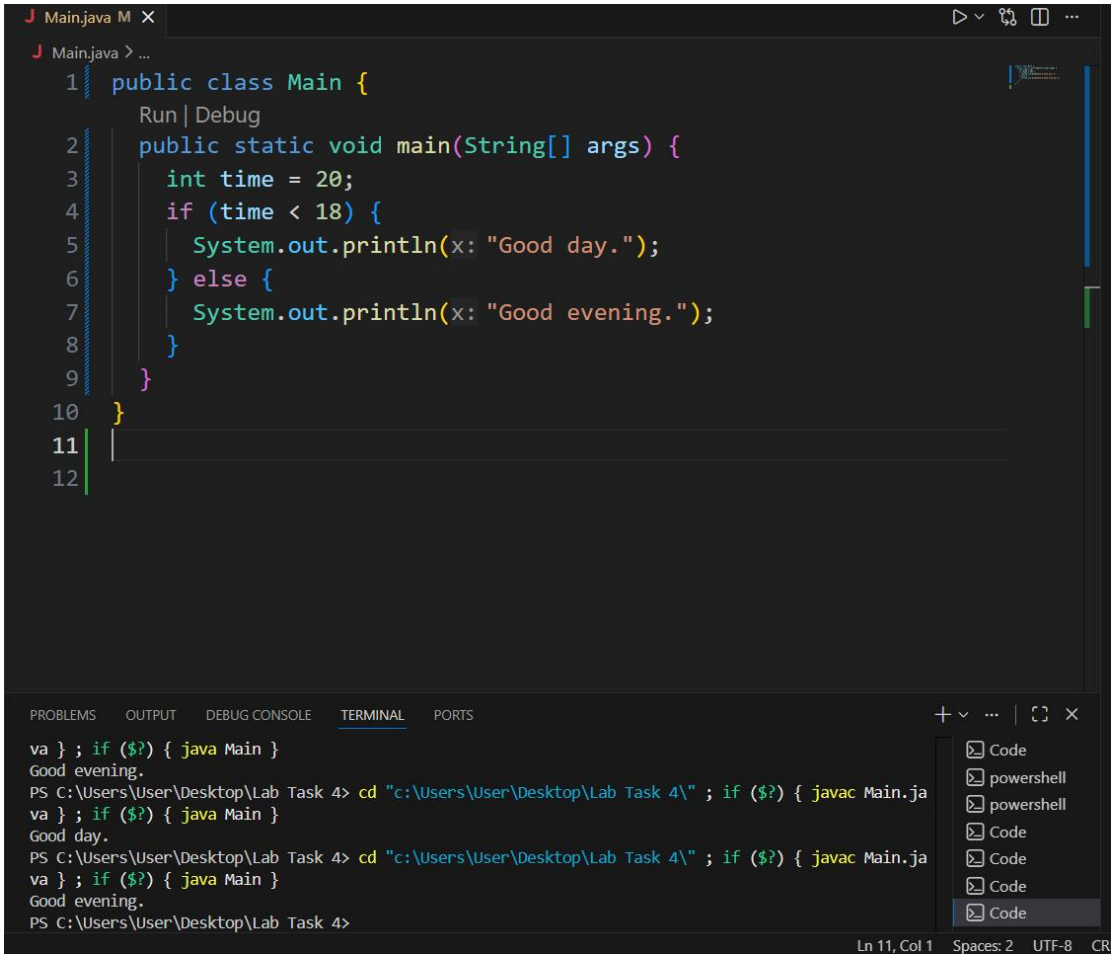


This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `time` is created with the value `14`. The program uses an `if-else if-else` statement to check different time conditions. If `time` is less than `10`, it prints `"Good morning."`. If `time` is less than `18`, it prints `"Good day."`. Otherwise, it prints `"Good evening."`. Since `time` is `14`, which is less than `18` but not less than `10`, the program prints `"Good day."`.

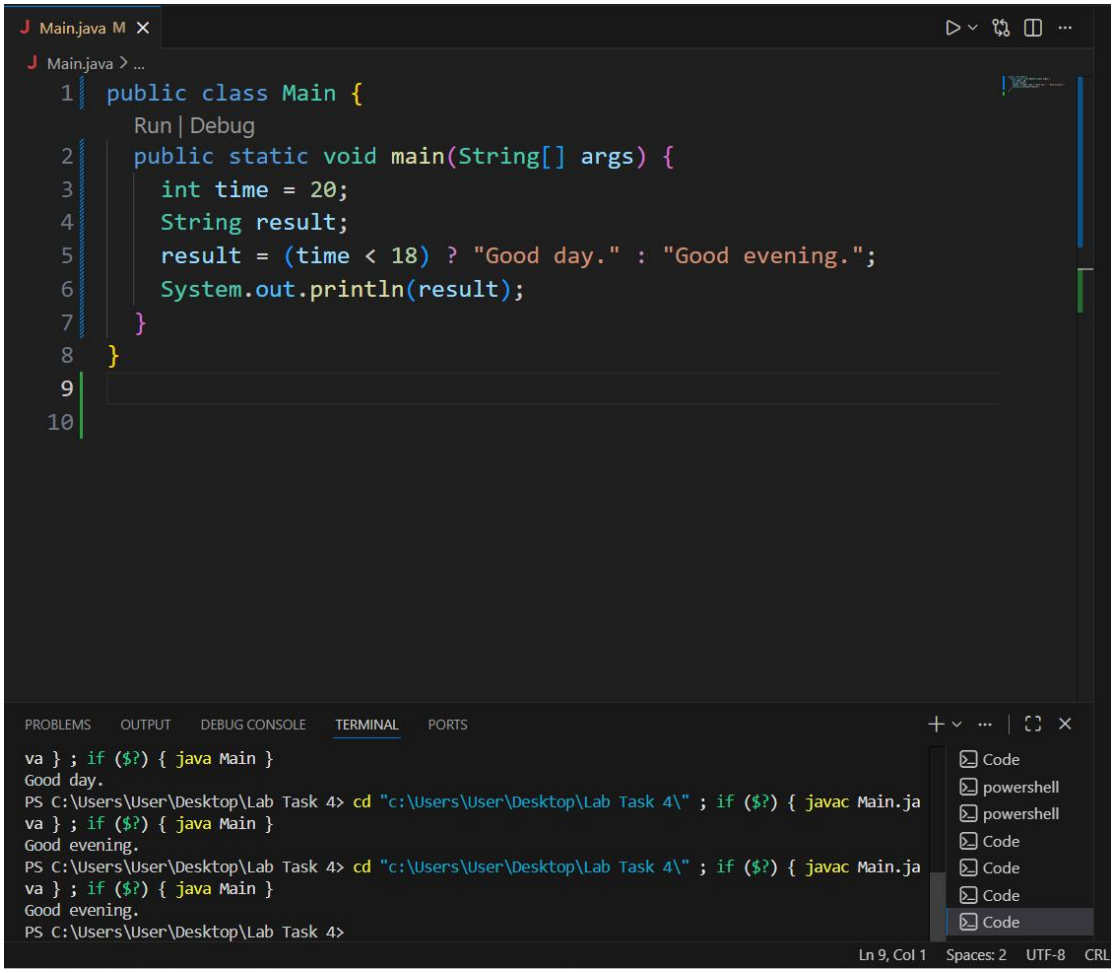


Short Hand IF ELSE

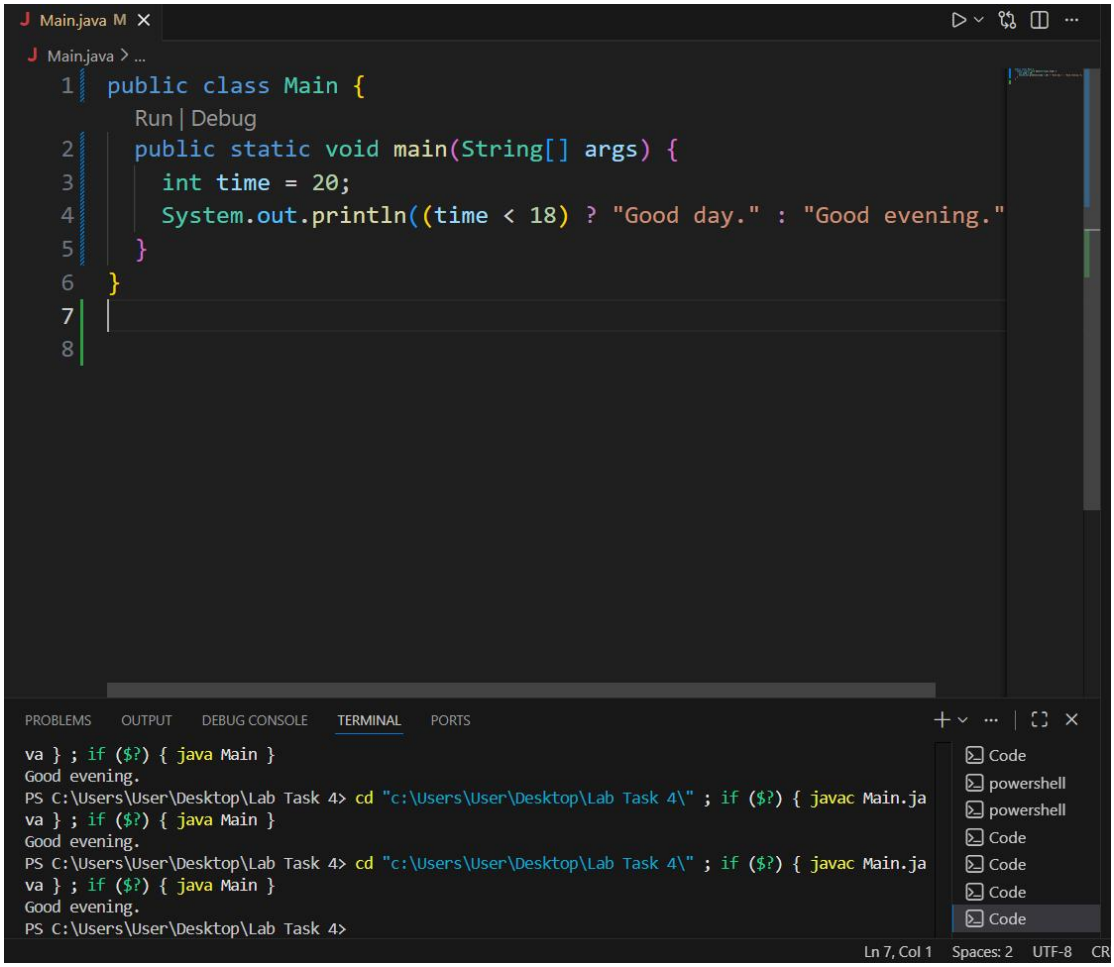
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, an integer variable time is created with the value 20. The program uses an if-else statement to check whether time is less than 18. If the condition is true, it prints "Good day."; otherwise, it prints "Good evening.". Since time is 20, which is greater than 18, the else block runs and "Good evening." is printed.



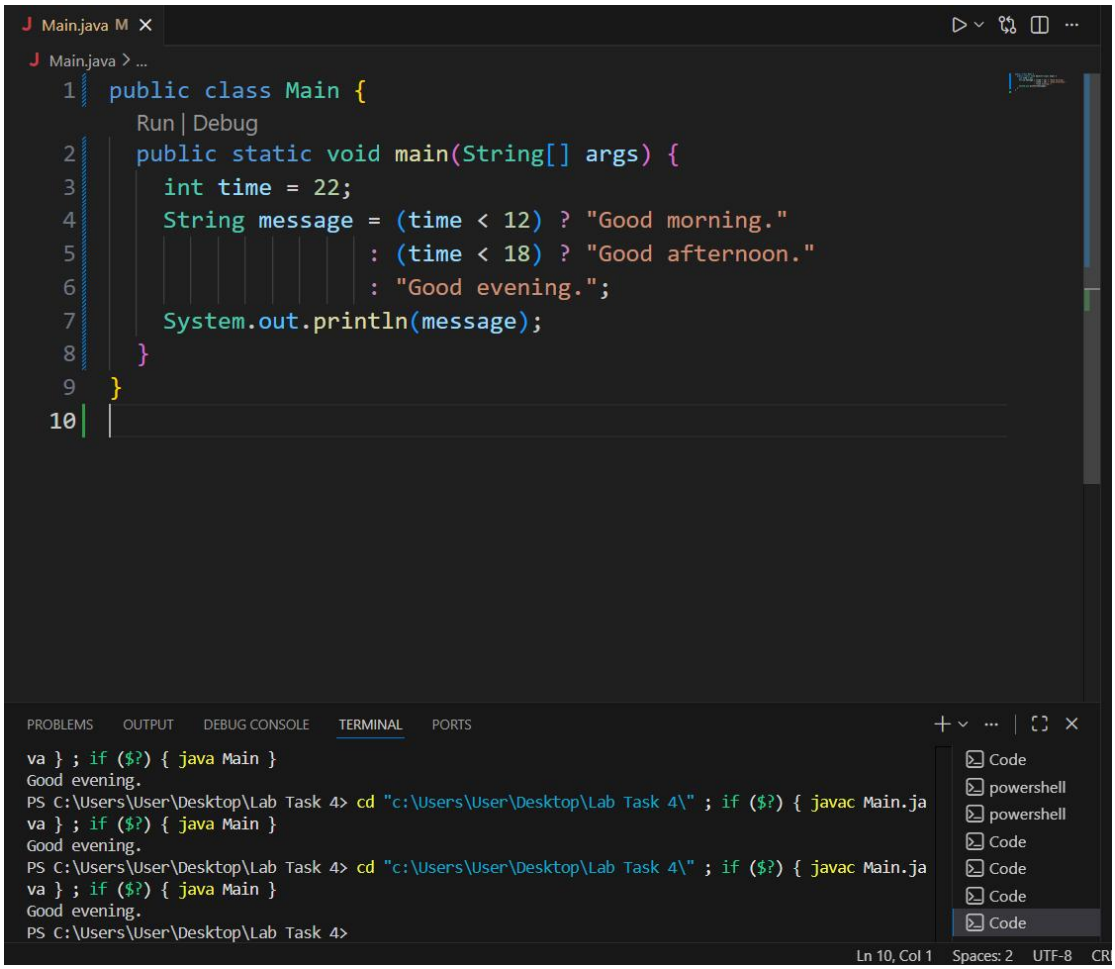
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `time` is created with the value `20`. A String variable `result` is then assigned using the ternary operator `(condition ? valueIfTrue : valueIfFalse)`. The condition `time < 18` is checked; if it is true, `"Good day."` is assigned to `result`, otherwise `"Good evening."` is assigned. Since `time` is `20`, the condition is false, so `"Good evening."` is printed on the screen.



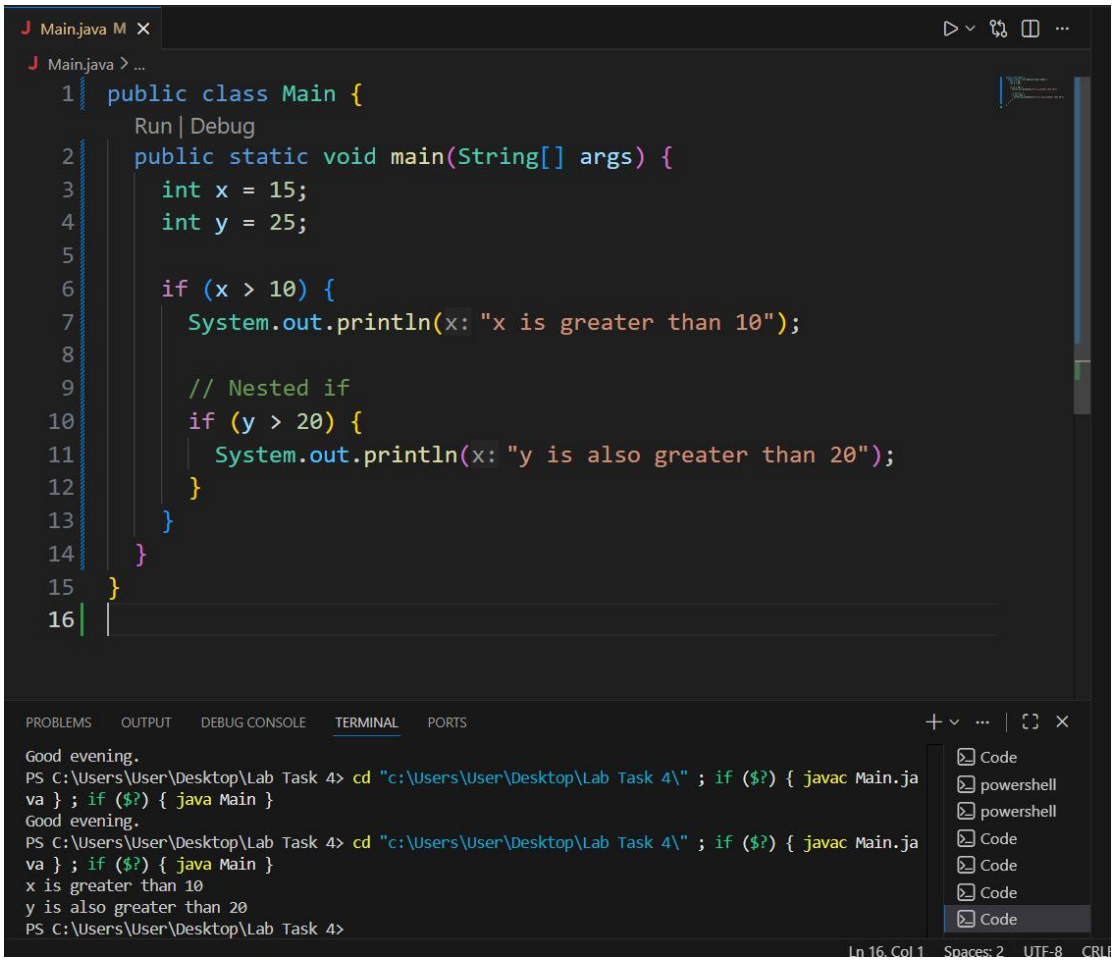
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `time` is created with the value `20`. The ternary operator is used directly inside `System.out.println` to check the condition `time < 18`. If the condition is true, `"Good day."` is printed; otherwise, `"Good evening."` is printed. Since `time` is `20`, the condition is false and the program prints `"Good evening."`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `time` is created with the value `22`. A String variable `message` is assigned using nested ternary operators to check different time ranges. If `time` is less than `12`, the message becomes `"Good morning."`. If it is less than `18`, the message becomes `"Good afternoon."`. Otherwise, the message becomes `"Good evening."`. Since `time` is `22`, the final message is `"Good evening."`, which is printed on the screen.



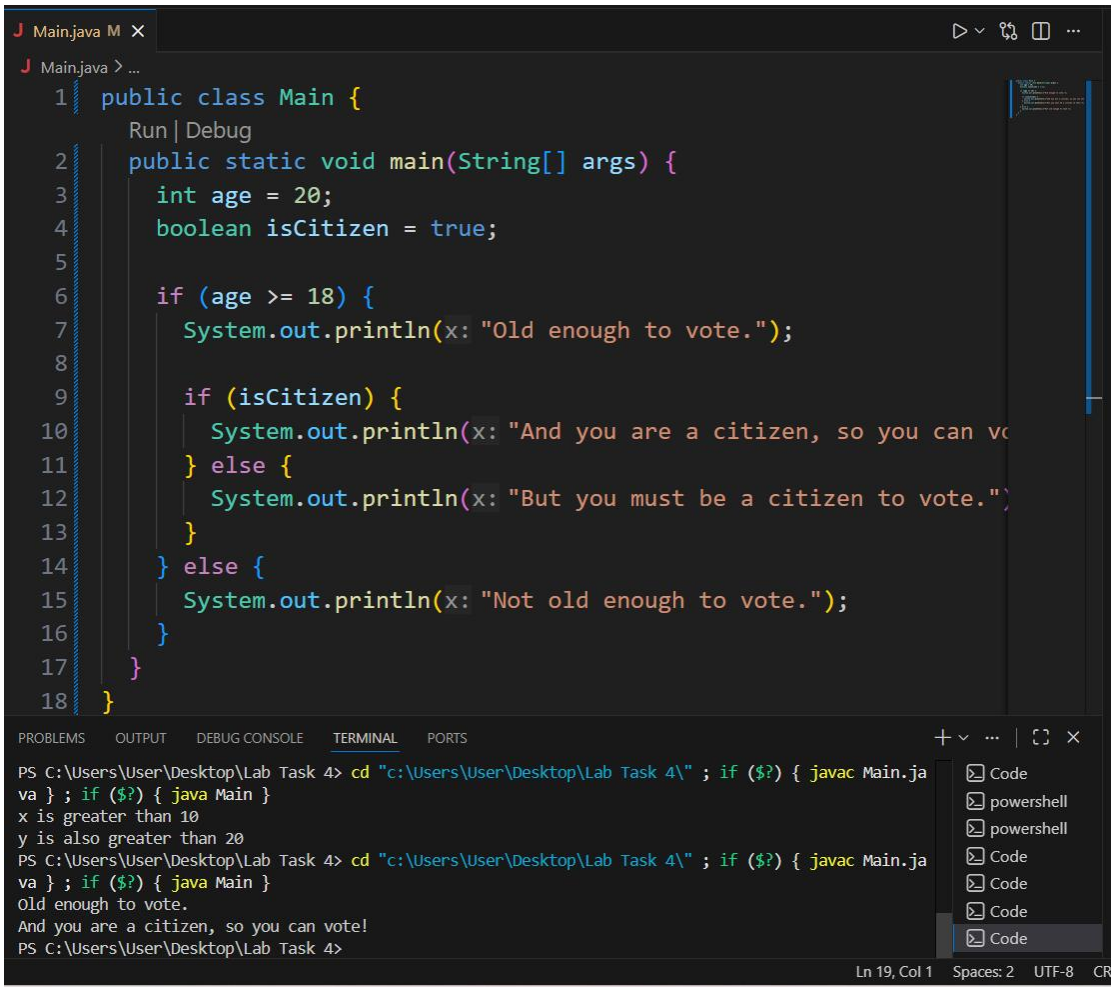
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two integer variables `x` and `y` are created with values `15` and `25`. The first `if` statement checks whether `x` is greater than `10`. Since this condition is true, the message `"x is greater than 10"` is printed. Inside this `if` block, there is a nested `if` statement that checks whether `y` is greater than `20`. Because `y` is also greater than `20`, the program prints `"y is also greater than 20"`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `age` is created with the value `20`, and a boolean variable `isCitizen` is set to `true`.

The program first checks if `age` is greater than or equal to `18`. Since `20` is greater than `18`, it prints `"Old enough to vote."`. Inside this `if` block, there is a nested `if-else` statement that checks whether `isCitizen` is `true`. Because `isCitizen` is `true`, it prints `"And you are a citizen, so you can vote!"`.

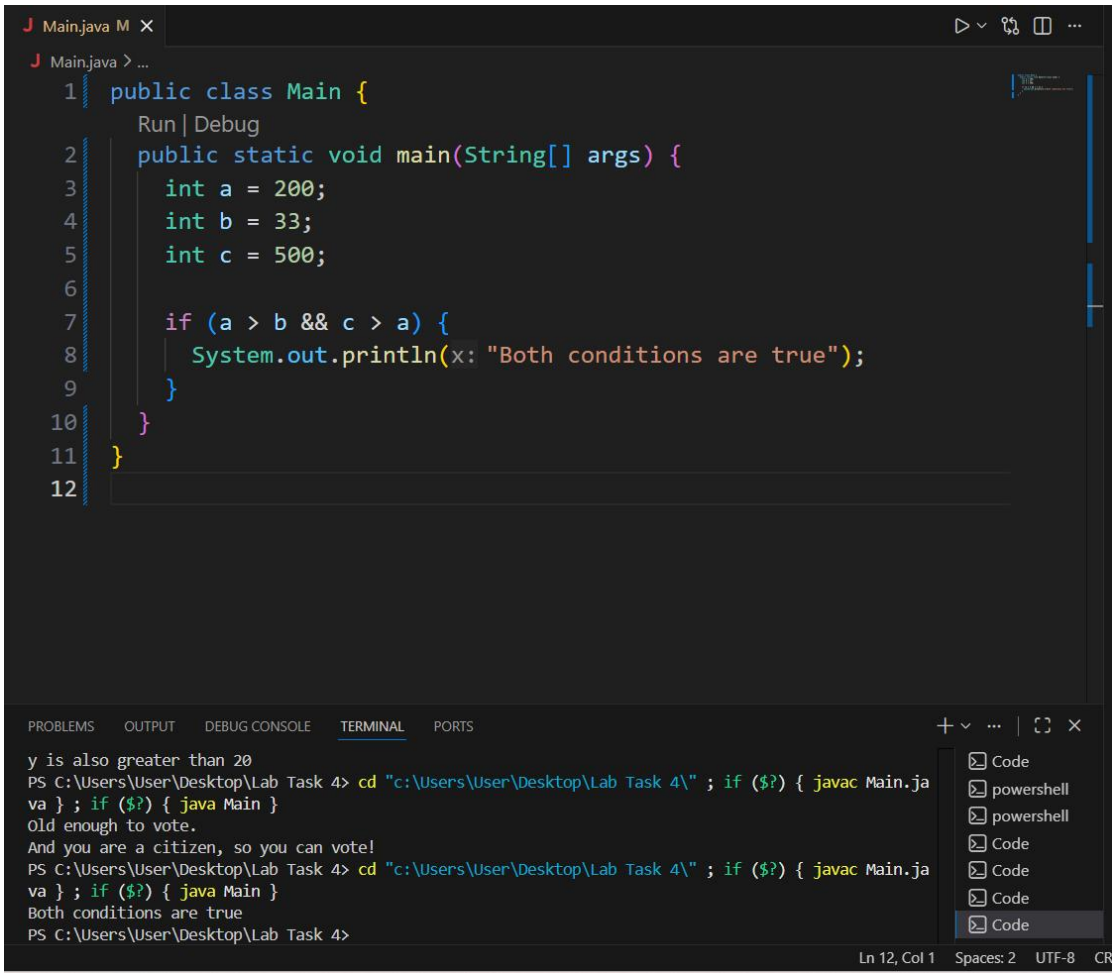
If `age` had been less than `18`, the outer `else` block would have printed `"Not old enough to vote."`. This shows how nested `if-else` statements can check multiple conditions step by step.



JAVA AND(&&)

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, three integer variables `a`, `b`, and `c` are created with values `200`, `33`, and `500`.

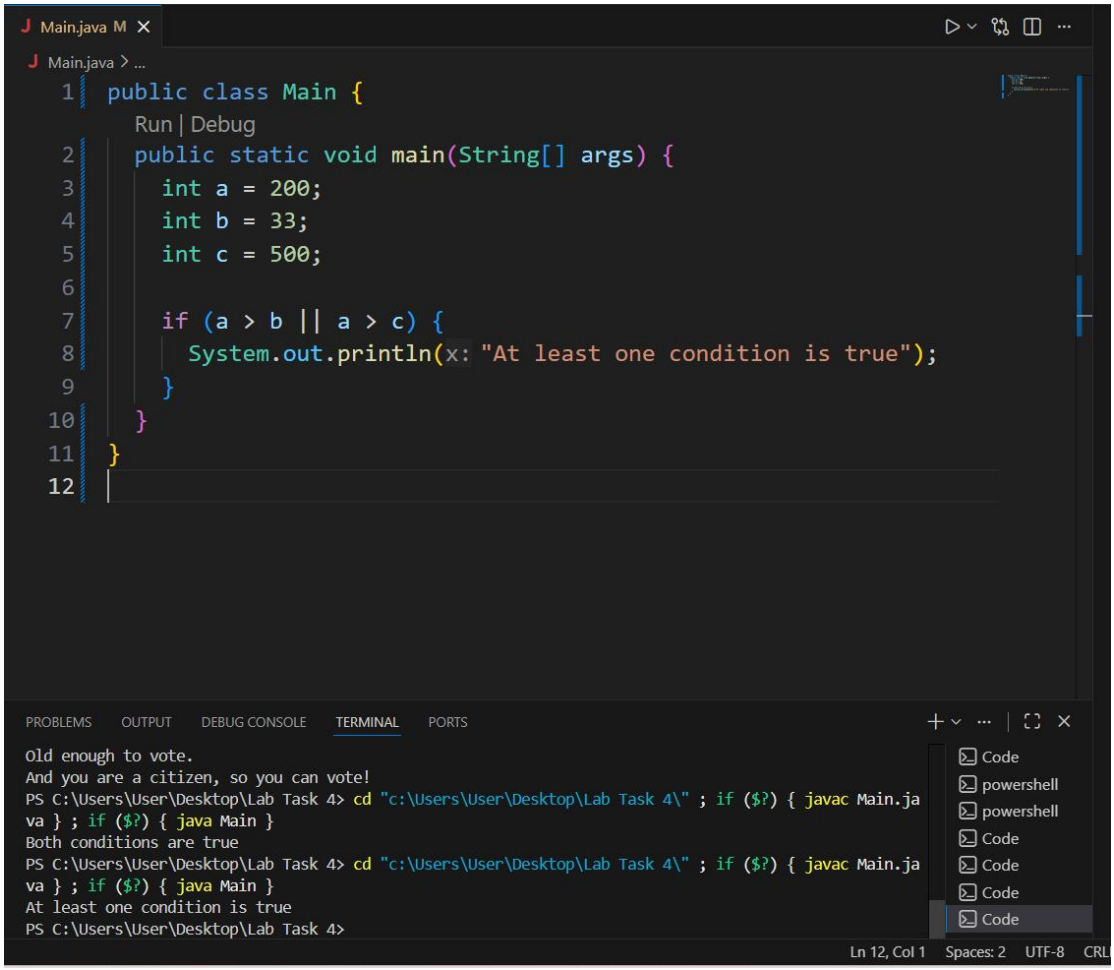
The `if` statement uses the logical AND operator `&&` to check two conditions at the same time: whether `a` is greater than `b` **and** whether `c` is greater than `a`. Since both conditions are true (`200 > 33` and `500 > 200`), the program prints `"Both conditions are true"` on the screen.



JAVA OR(||)

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, three integer variables `a`, `b`, and `c` are created with values `200`, `33`, and `500`.

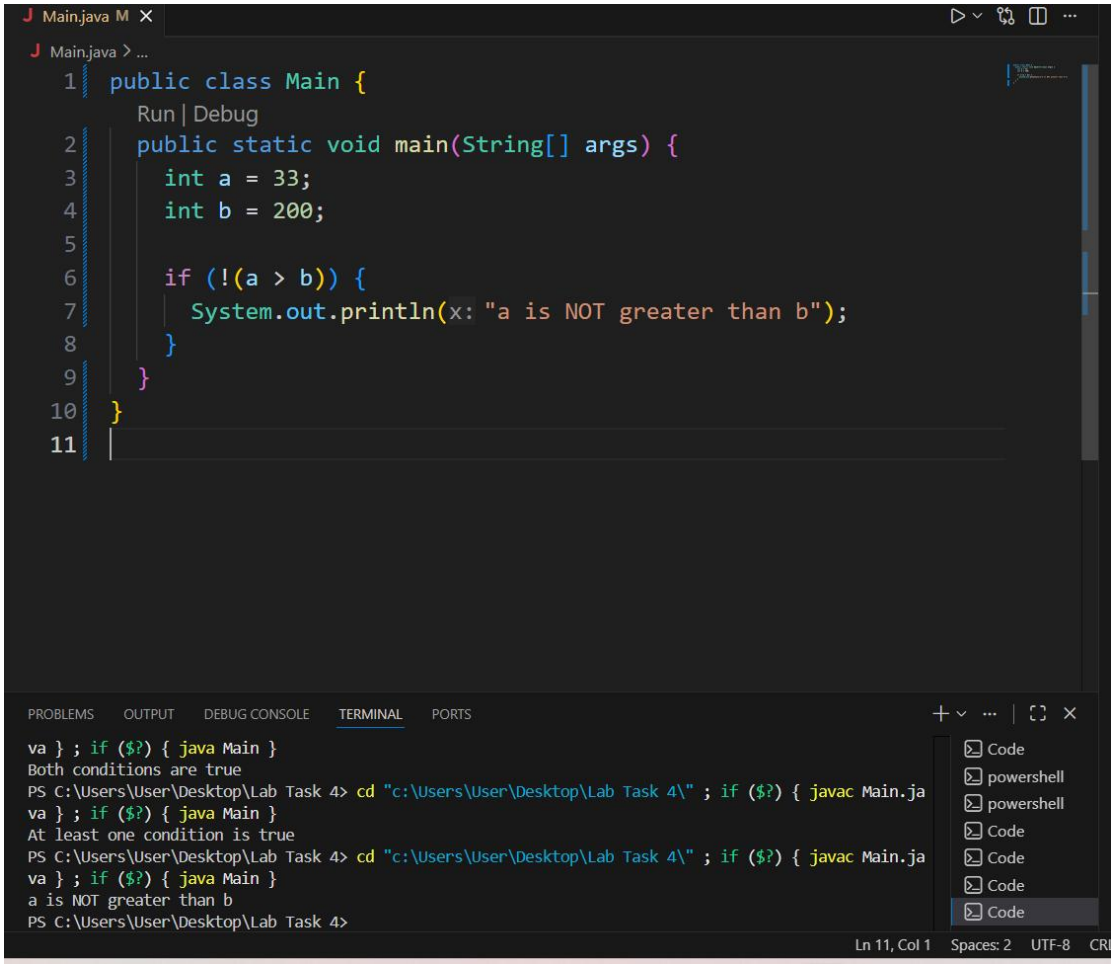
The `if` statement uses the logical OR operator `||` to check if **at least one** of two conditions is true: whether `a` is greater than `b` **or** whether `a` is greater than `c`. Since `a > b` is true (`200 > 33`), the condition is satisfied, and the program prints `"At least one condition is true"` on the screen.



JAVA NOT(!)

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two integer variables `a` and `b` are created with values `33` and `200`.

The `if` statement uses the logical NOT operator `!` to reverse the result of the condition `a > b`. Since `a > b` is false (`33` is not greater than `200`), the `!` operator makes the condition true. Therefore, the program prints `"a is NOT greater than b"` on the screen.

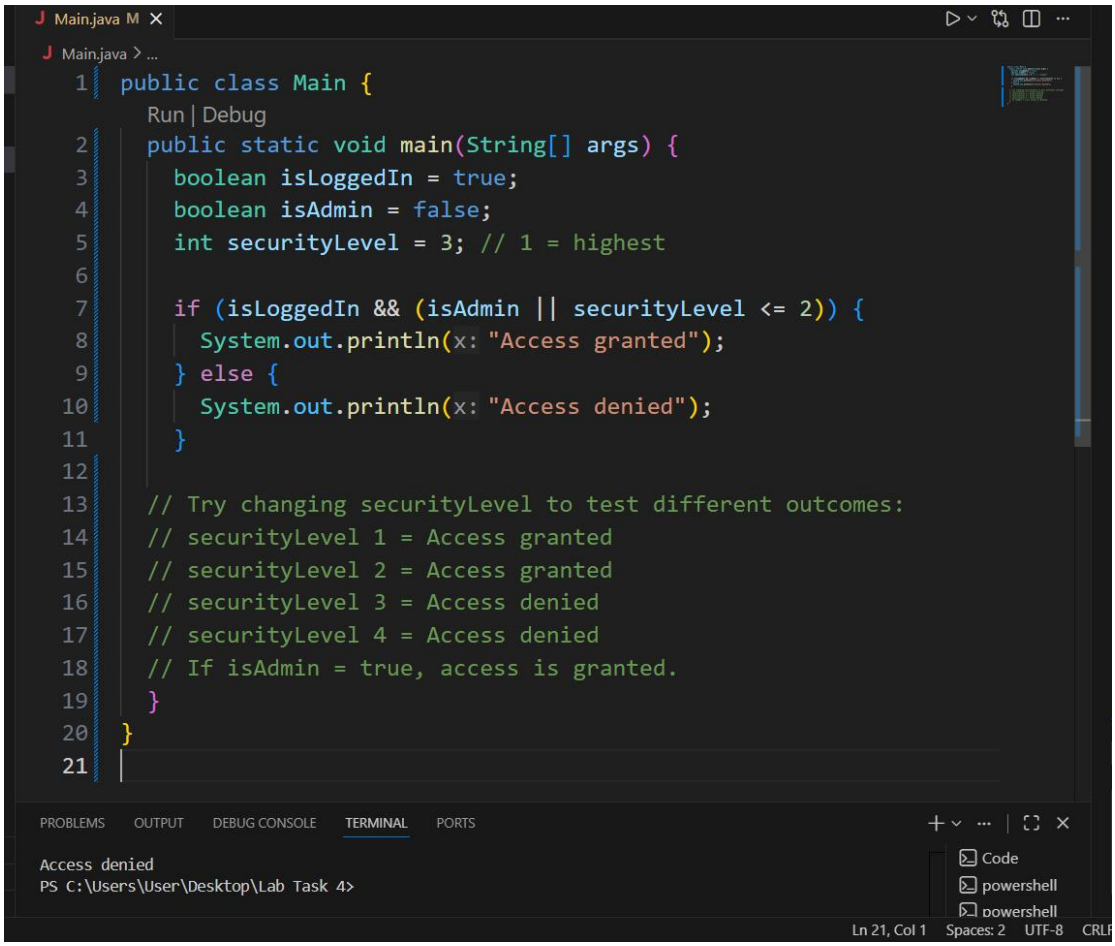


Real Life Example

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two boolean variables `isLoggedIn` and `isAdmin` are created, along with an integer `securityLevel`.

The `if` statement checks whether the user is logged in **and** either is an admin **or** has a high enough security level (securityLevel ≤ 2). If this combined condition is true, the program prints `"Access granted"`; otherwise, it prints `"Access denied"`.

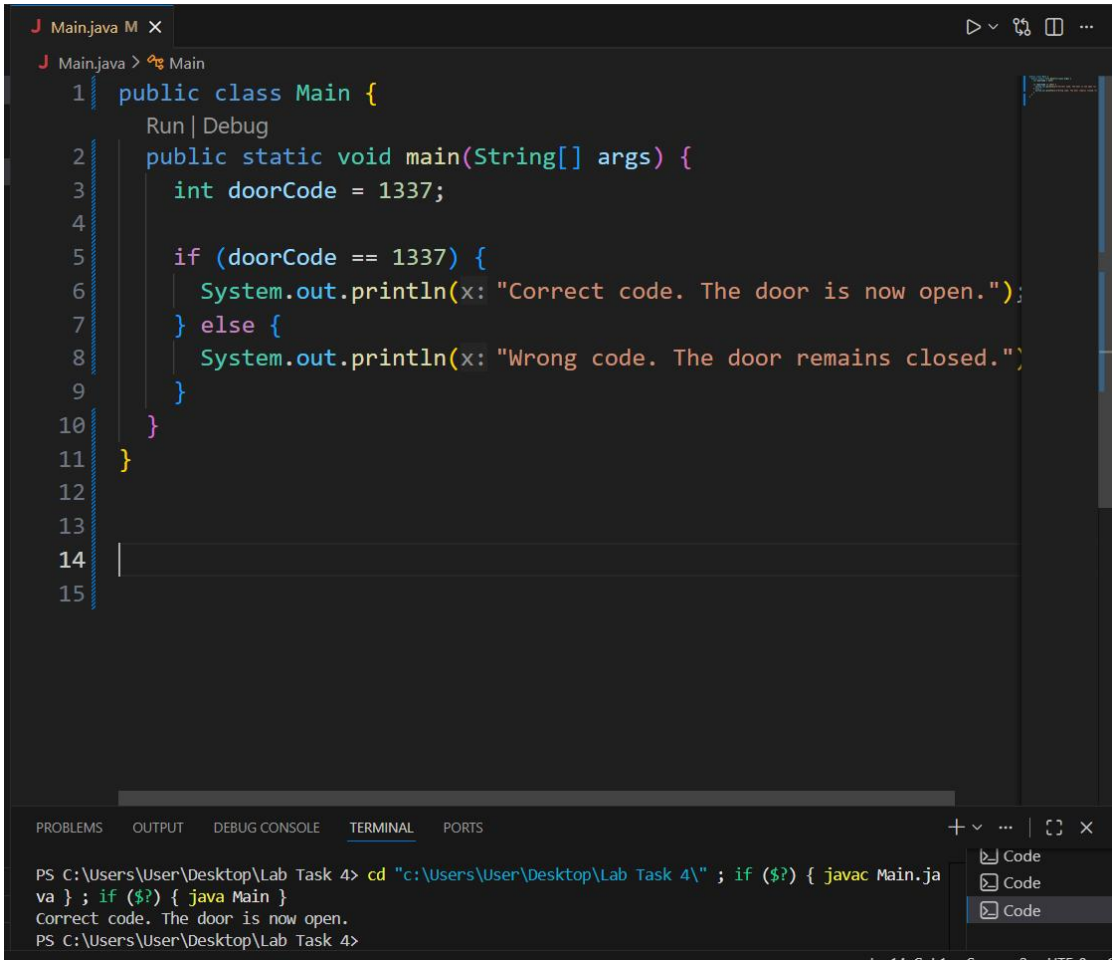
In this example, `isLoggedIn` is `true`, `isAdmin` is `false`, and `securityLevel` is `3`. Since the security level is higher than 2 and the user is not an admin, the condition is false, so `"Access denied"` is printed. The comments explain how changing `securityLevel` or `isAdmin` affects access.



JAVA Real Life Examples

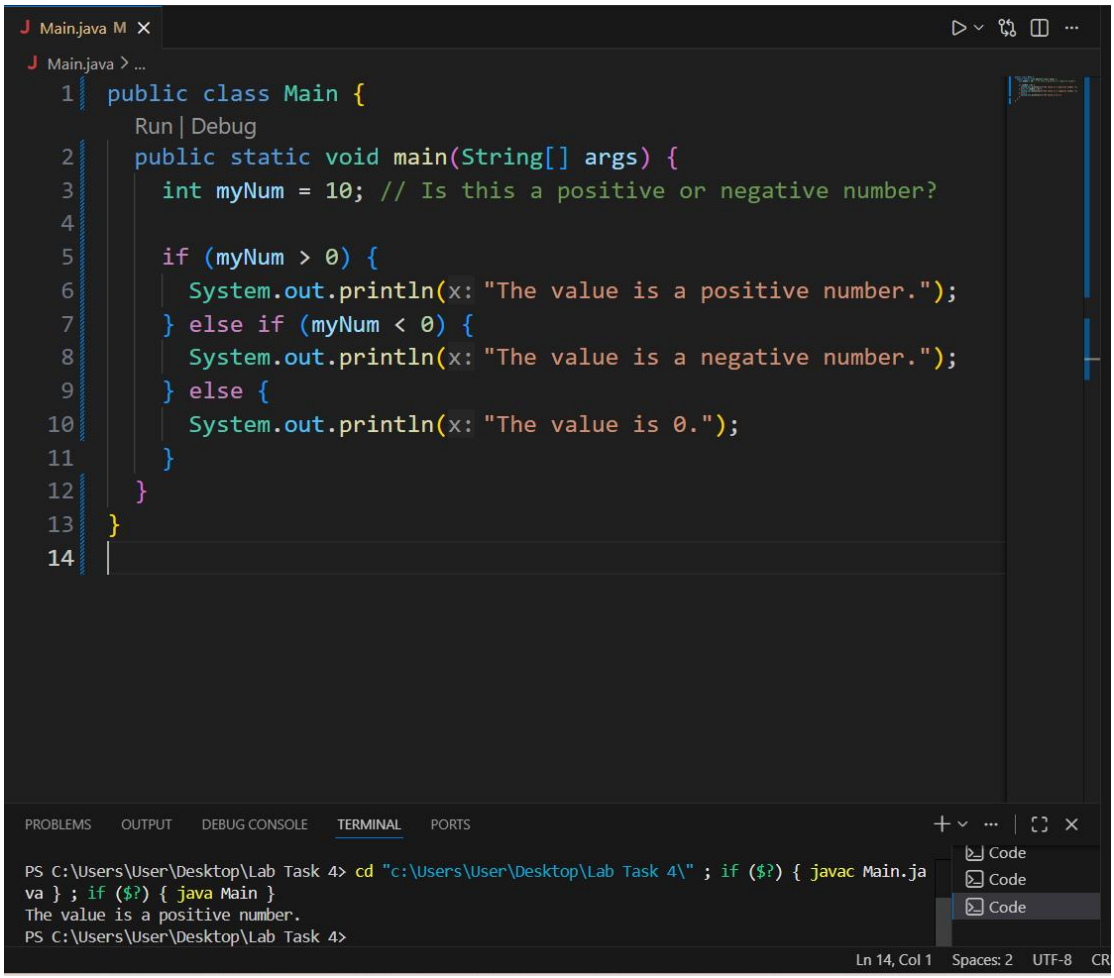
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `doorCode` is created with the value `1337`.

The program uses an `if-else` statement to check whether `doorCode` is equal to `1337`. If it is, the program prints `"Correct code. The door is now open."`. Otherwise, it prints `"Wrong code. The door remains closed."`. Since `doorCode` is `1337`, the condition is true and the first message is printed.



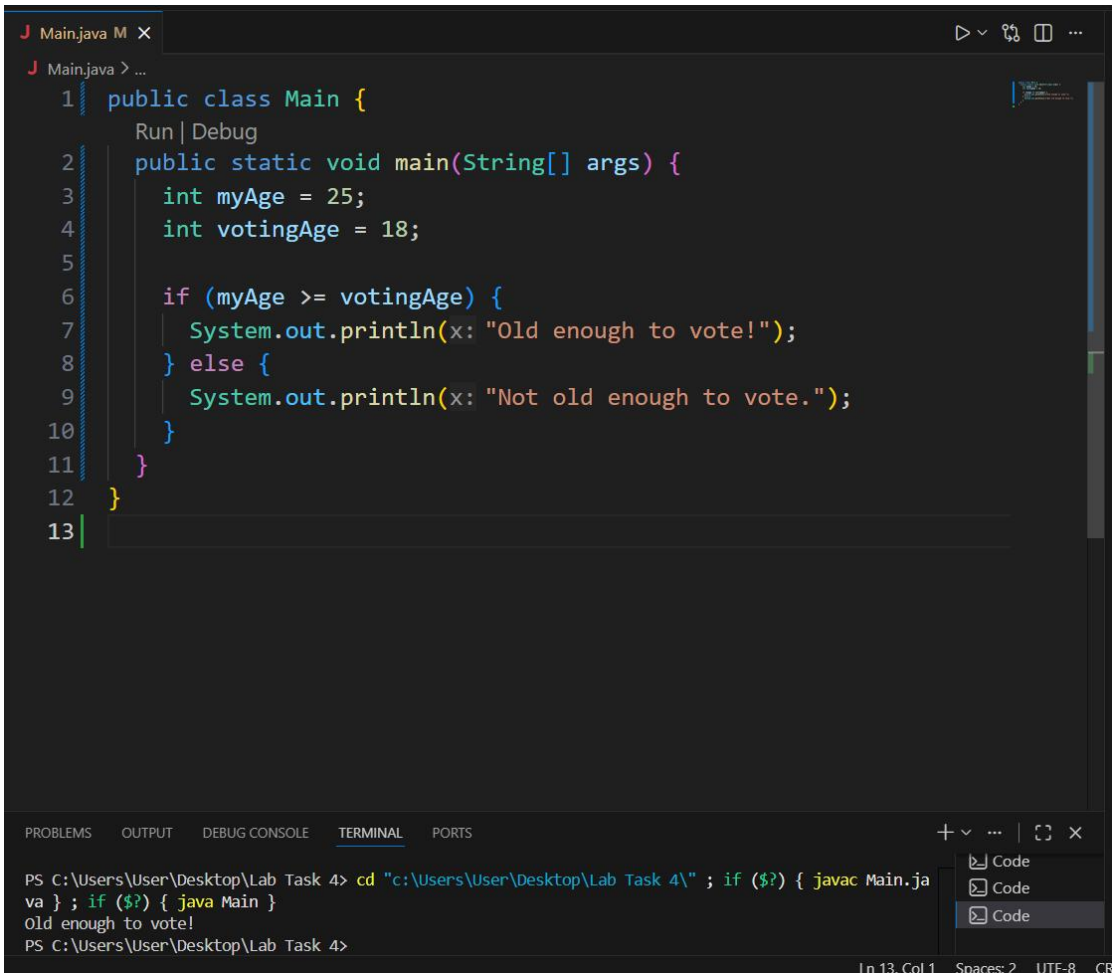
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `myNum` is created with the value `10`.

The program uses an `if-else if-else` statement to check whether the number is positive, negative, or zero. If `myNum > 0`, it prints `"The value is a positive number."`. If `myNum < 0`, it prints `"The value is a negative number."`. Otherwise, it prints `"The value is 0."`. Since `myNum` is `10`, the first condition is true, so `"The value is a positive number."` is printed.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two integer variables `myAge` and `votingAge` are created with values `25` and `18`.

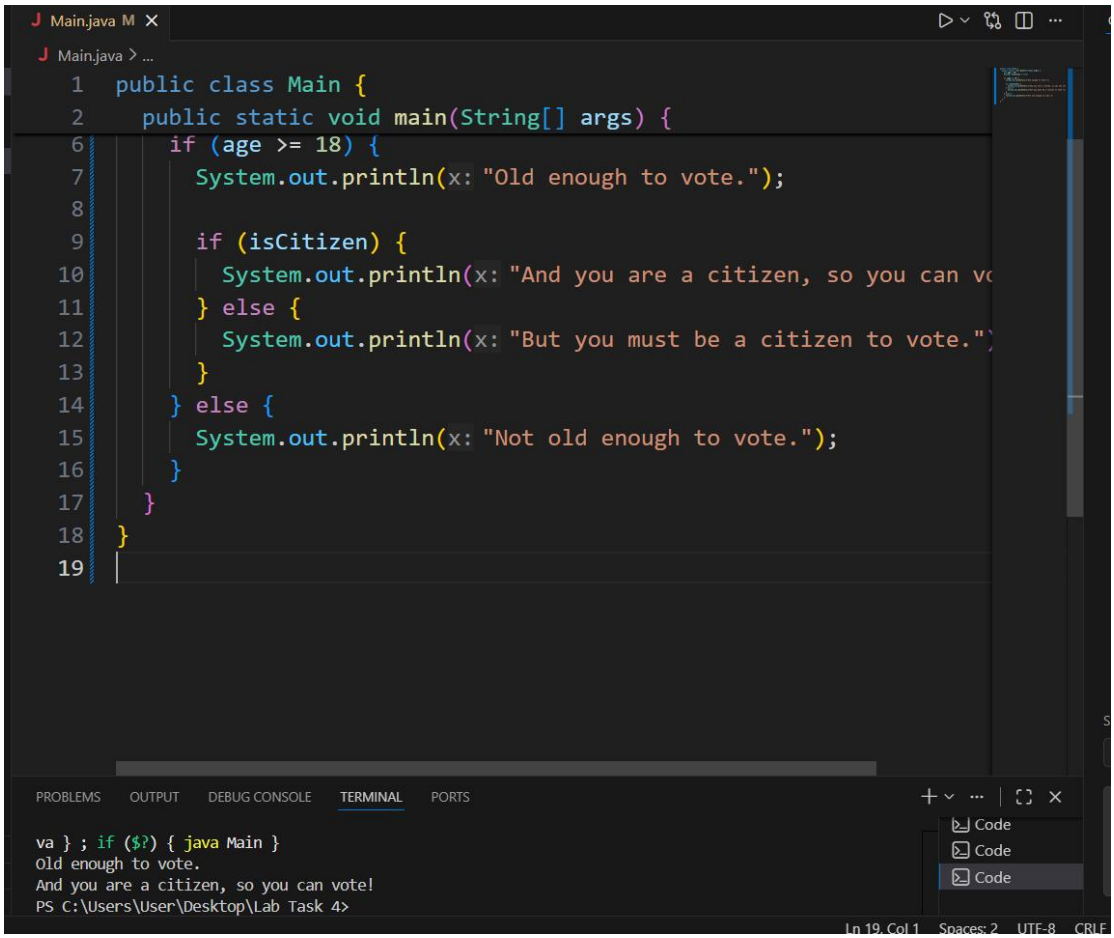
The program uses an `if-else` statement to check whether `myAge` is greater than or equal to `votingAge`. If the condition is true, it prints `"Old enough to vote!"`; otherwise, it prints `"Not old enough to vote."`. Since `myAge` is `25`, which is greater than `18`, the program prints `"Old enough to vote!"`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `age` is set to `20`, and a boolean variable `isCitizen` is set to `true`.

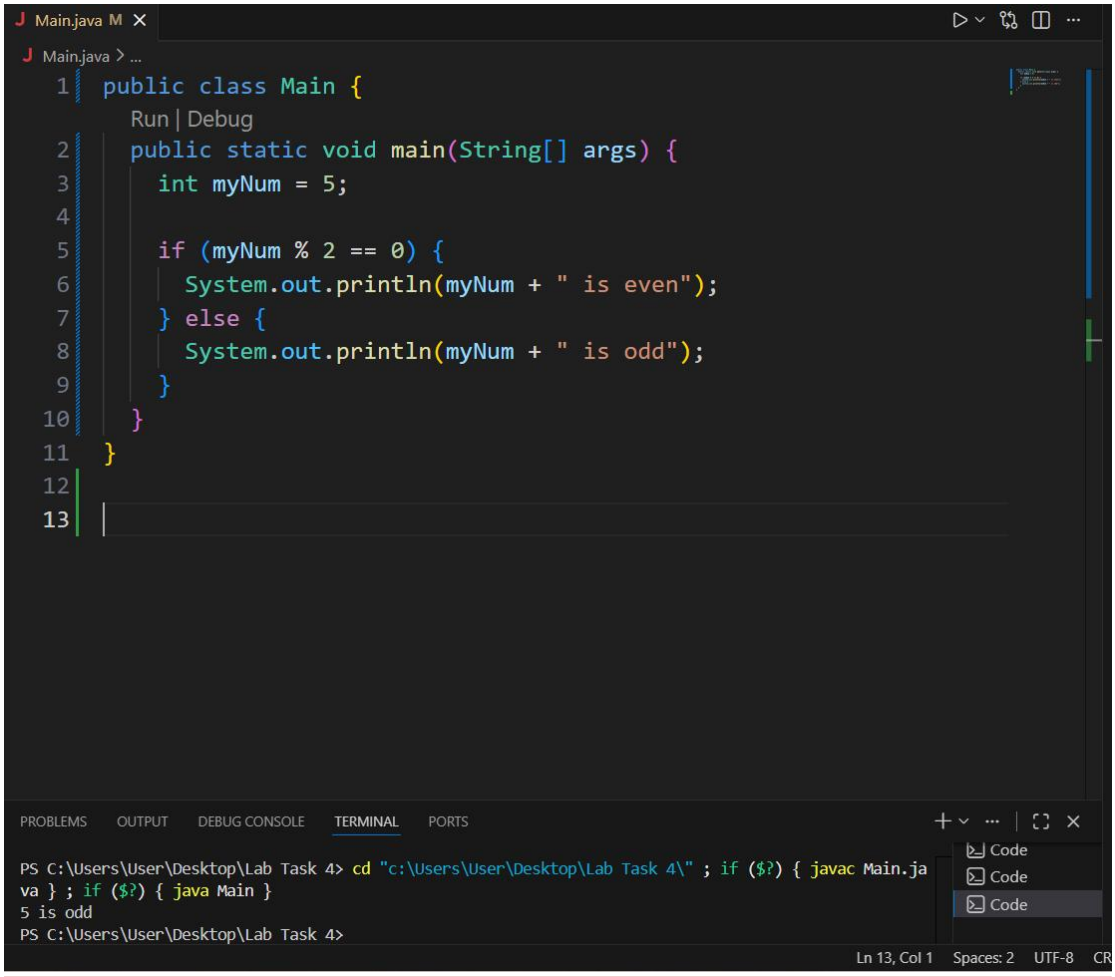
The program first checks if `age` is greater than or equal to `18`. Since `20` is greater than `18`, it prints `"Old enough to vote."`. Then, inside this `if` block, there is a nested `if-else` statement that checks whether `isCitizen` is `true`. Because `isCitizen` is `true`, it prints `"And you are a citizen, so you can vote!"`.

If `age` had been less than `18`, the outer `else` block would have printed `"Not old enough to vote."`. This shows how nested `if-else` statements can handle multiple conditions step by step.



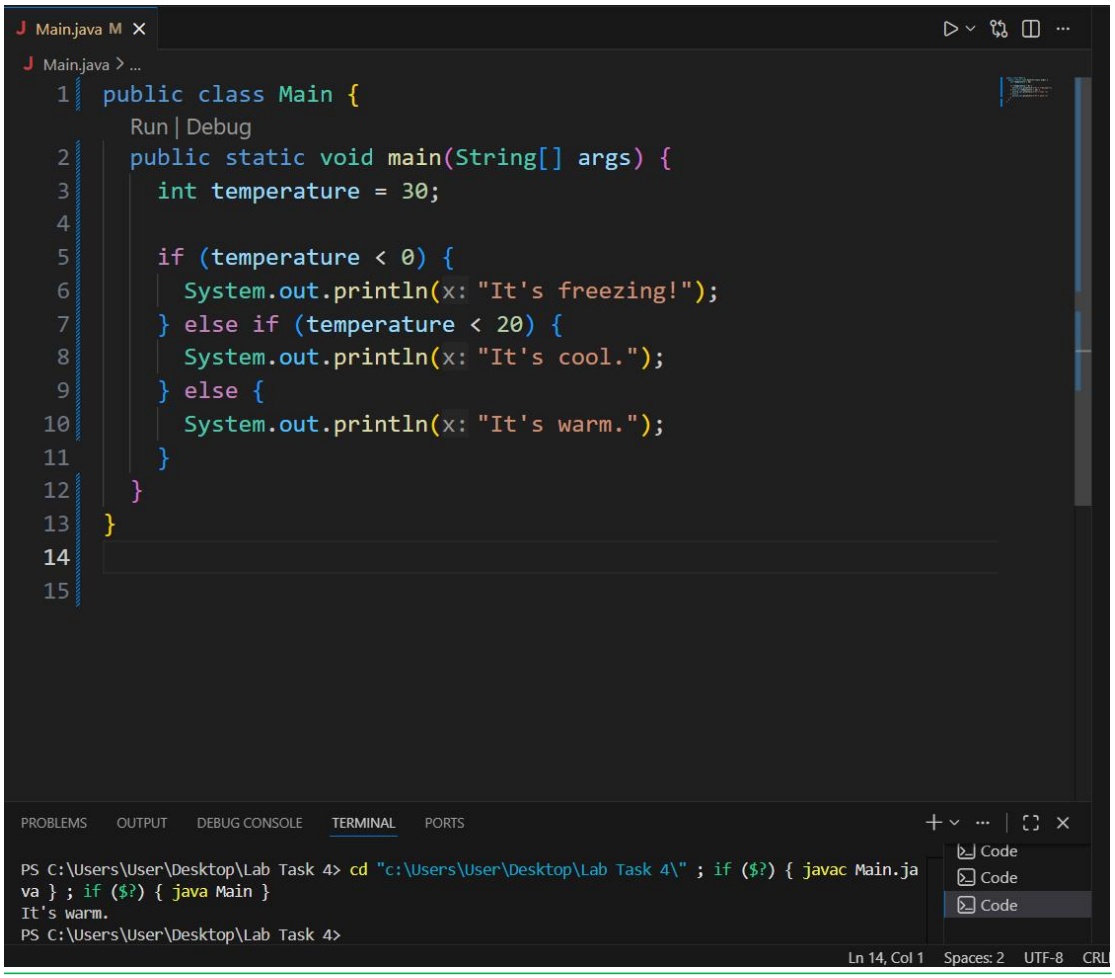
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `myNum` is created with the value `5`.

The program uses an `if-else` statement to check whether the number is even or odd. It does this by using the modulus operator `%` to find the remainder when `myNum` is divided by `2`. If the remainder is `0`, the number is even and it prints `"5 is even"`. Otherwise, the number is odd and it prints `"5 is odd"`. Since `5 % 2` is `1`, the program prints `"5 is odd"`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `temperature` is created with the value `30`.

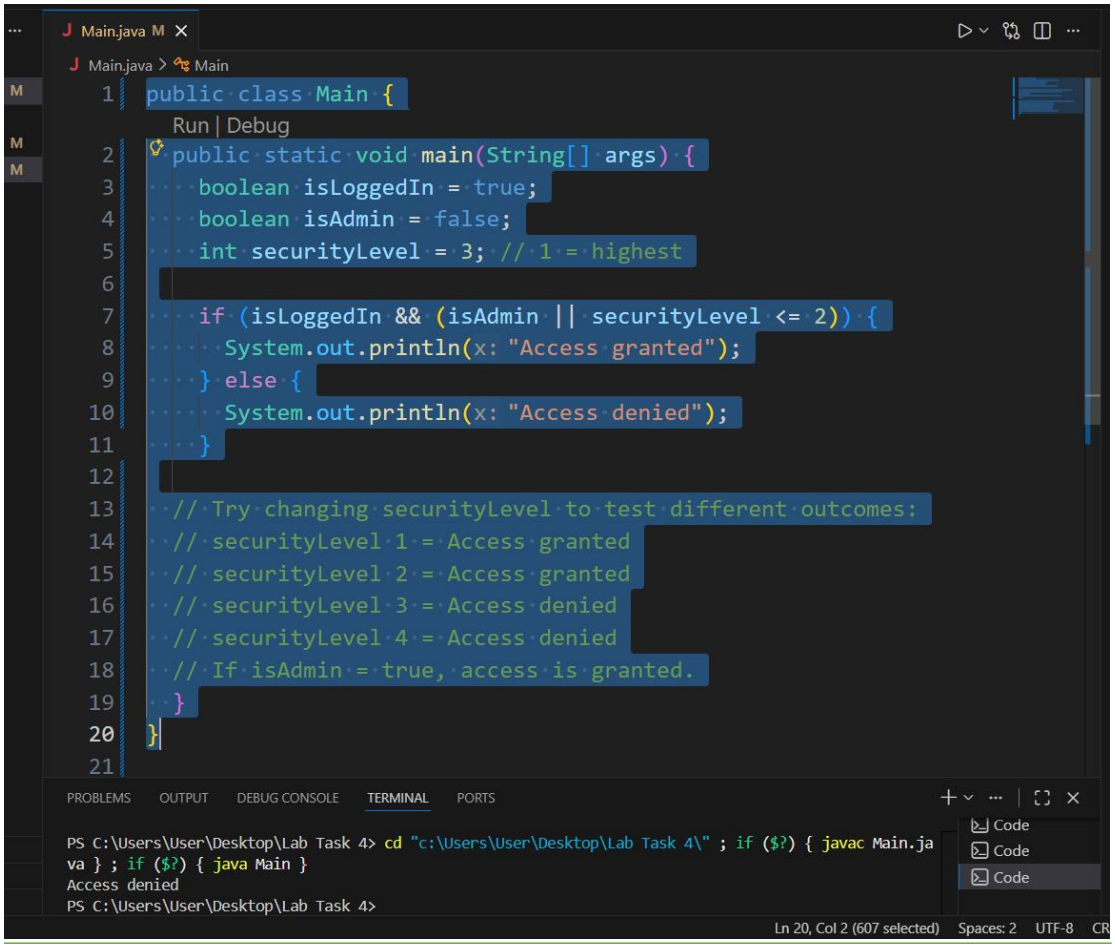
The program uses an `if-else if-else` statement to check the temperature. If `



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, two boolean variables `isLoggedIn` and `isAdmin` are created, and an integer variable `securityLevel` is set to `3`.

The `if` statement checks a combination of conditions: the user must be logged in **and** either be an admin **or** have a security level less than or equal to 2. If this condition is true, the program prints `"Access granted"`; otherwise, it prints `"Access denied"`.

In this example, `isLoggedIn` is `true`, `isAdmin` is `false`, and `securityLevel` is `3`. Since the user is not an admin and the security level is higher than 2, the condition is false, so the program prints `"Access denied"`. The comments explain how changing `securityLevel` or `isAdmin` affects the outcome.

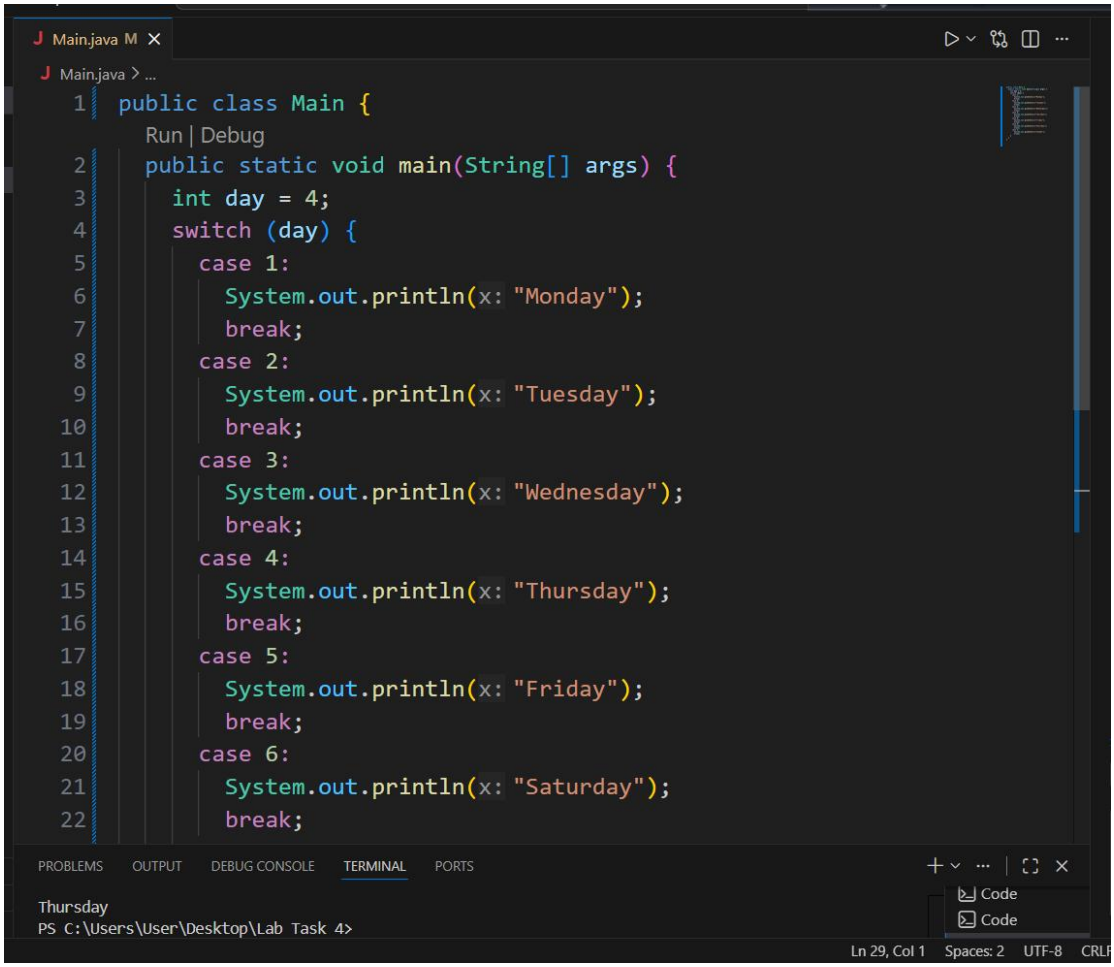


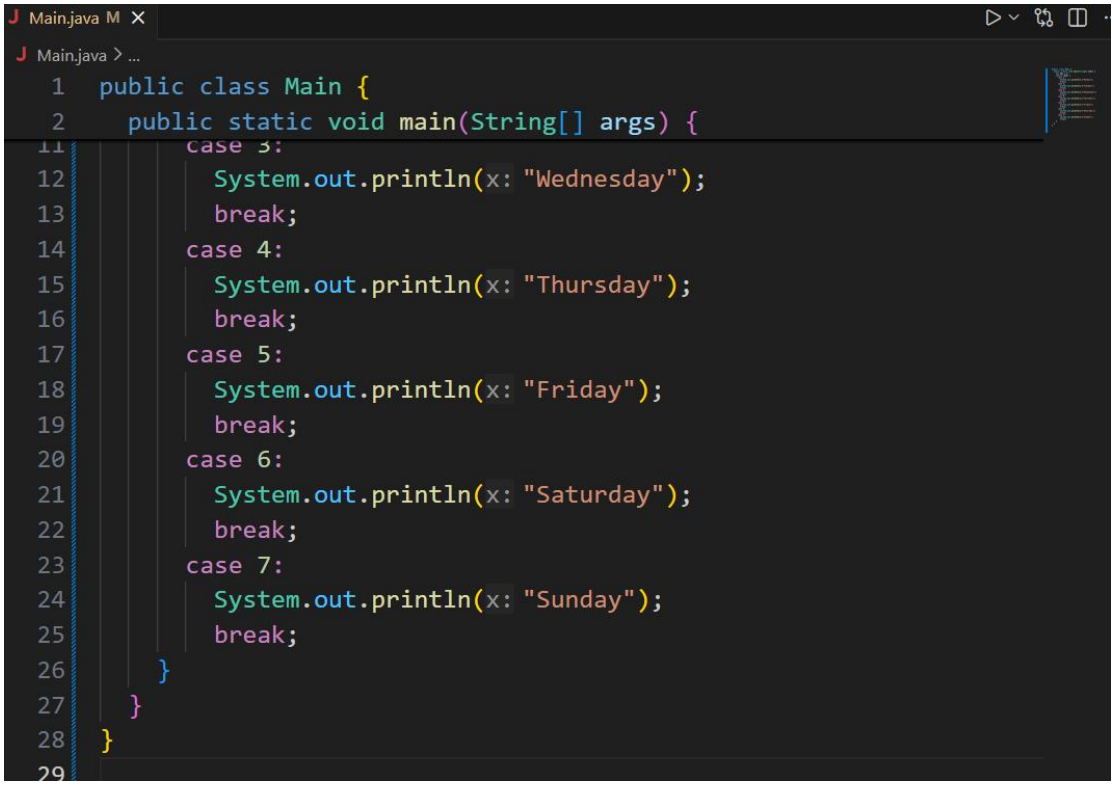
JAVA Switch

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `day` is created with the value `4`.

The program uses a `switch` statement to match the value of `day` with different cases. Each `case` represents a day of the week. When `day` matches a case, the corresponding message is printed, and the `break` statement stops the program from checking the remaining cases.

Since `day` is `4`, the program matches `case 4` and prints `"Thursday"` on the screen.

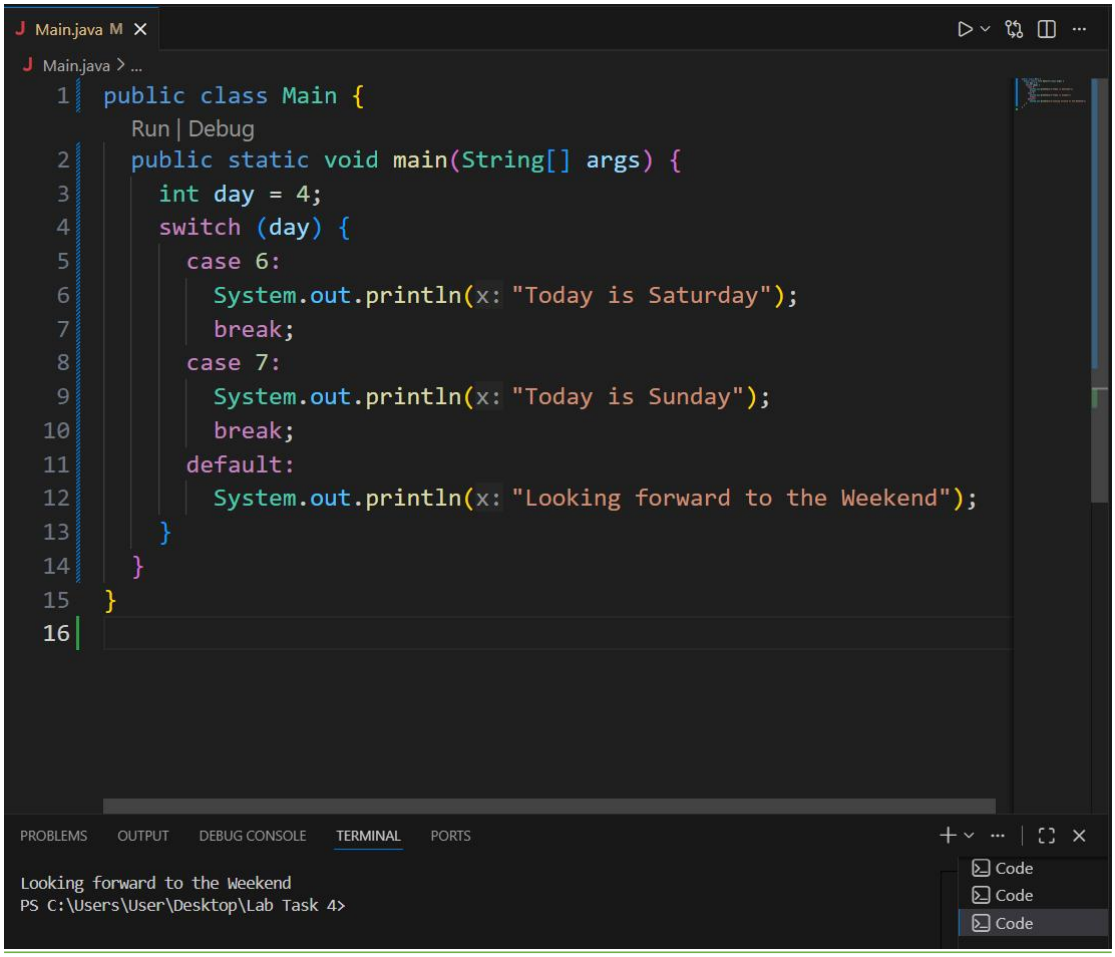



Main.java M
Main.java > ...
public class Main {
public static void main(String[] args) {
case 3:
System.out.println(x: "Wednesday");
break;
case 4:
System.out.println(x: "Thursday");
break;
case 5:
System.out.println(x: "Friday");
break;
case 6:
System.out.println(x: "Saturday");
break;
case 7:
System.out.println(x: "Sunday");
break;
}
}
}

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `day` is created with the value `4`.

The program uses a `switch` statement to check the value of `day`. It has two specific cases: `6` for Saturday and `7` for Sunday. If `day` matches one of these cases, the corresponding message is printed. If `day` does not match any case, the `default` block runs.

Since `day` is `4`, which does not match `6` or `7`, the program executes the `default` block and prints `"Looking forward to the Weekend"`.

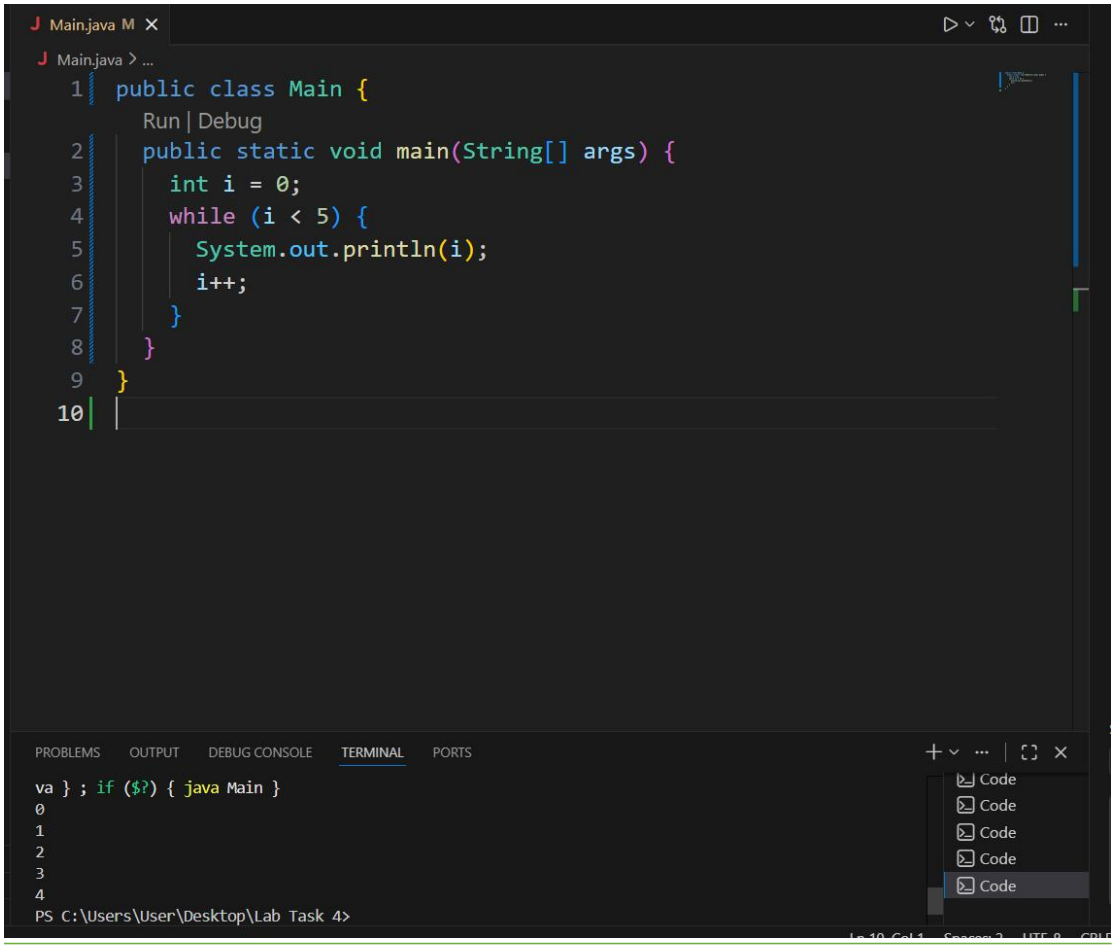


JAVA For Loop

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `i` is created and set to `0`.

The program uses a `while` loop that continues as long as `i` is less than `5`. Inside the loop, it prints the current value of `i` and then increases `i` by `1` using `i++`.

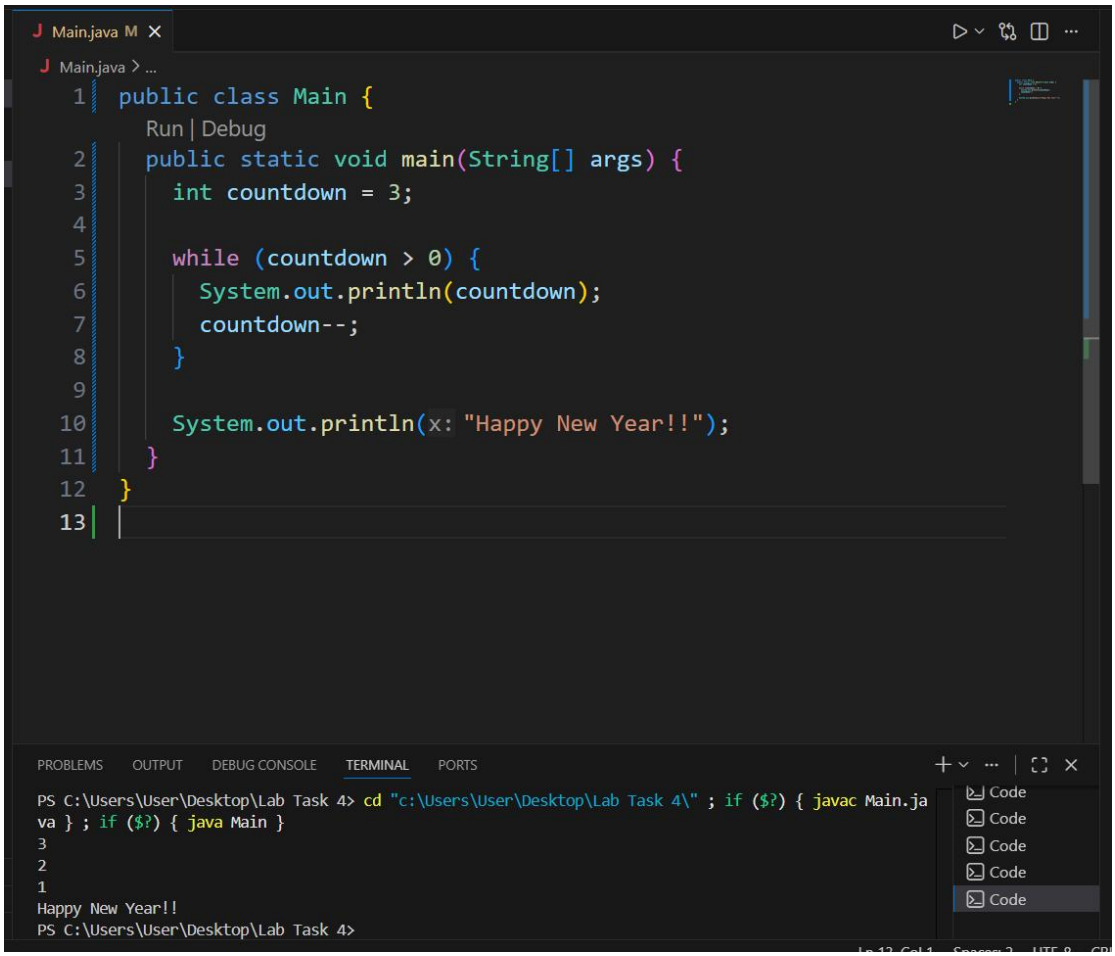
This repeats until `i` reaches `5`, at which point the condition `i < 5` becomes false and the loop stops.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `countdown` is created and set to `3`.

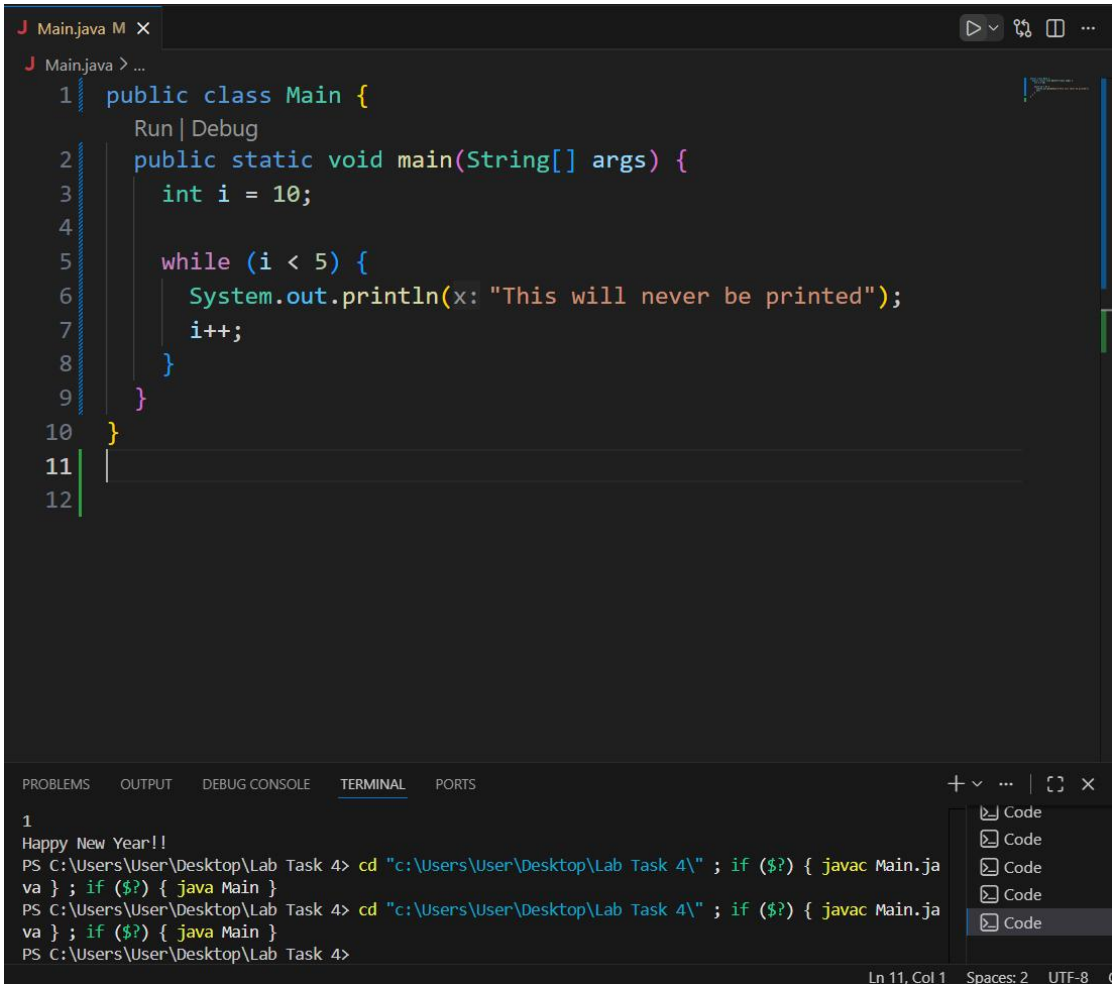
The program uses a `while` loop that continues as long as `countdown` is greater than `0`. Inside the loop, it prints the current value of `countdown` and then decreases `countdown` by `1` using `countdown--`.

Once `countdown` reaches `0`, the loop stops, and the program prints `"Happy New Year!!"`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `i` is created and set to `10`.

The program uses a `while` loop with the condition `i < 5`. Since `i` is already `10`, which is not less than `5`, the condition is false from the start. Therefore, the loop body never runs, and nothing is printed on the screen. This shows that a `while` loop only executes when its condition is true.

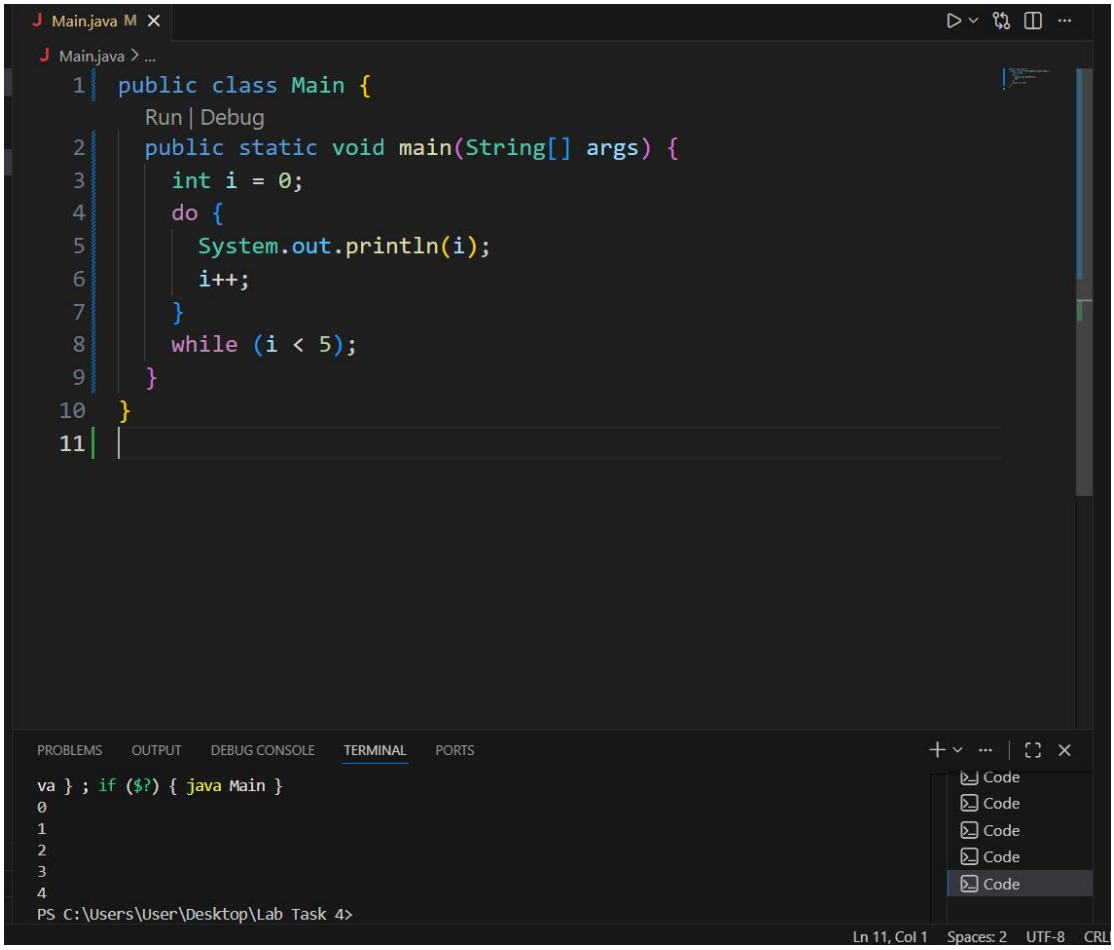


JAVA Do/While Loop

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `i` is created and set to `0`.

The program uses a `do-while` loop, which executes the code inside the `do` block **at least once** before checking the condition. Inside the loop, it prints the current value of `i` and then increases `i` by `1` using `i++`.

The loop continues as long as `i < 5`.

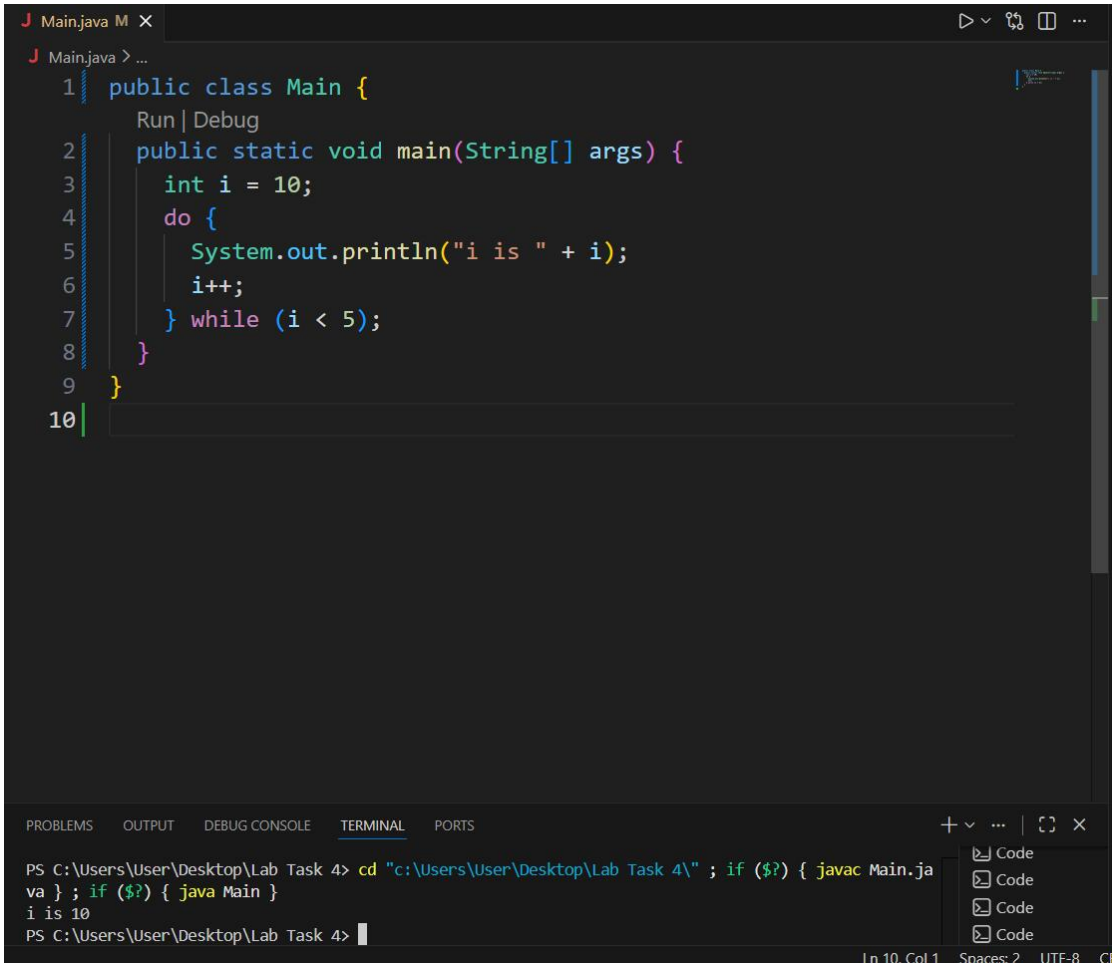


This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `i` is created and set to `10`.

The program uses a `do-while` loop. The code inside the `do` block runs **at least once** before the condition is checked. Inside the loop, it prints `"i is 10"` and then increases `i` by `1`.

After that, the condition `i < 5` is checked. Since `i` is now `11`, the condition is false, so the loop stops.

This shows that a `do-while` loop always executes its body at least once, even if the condition is false from the start.

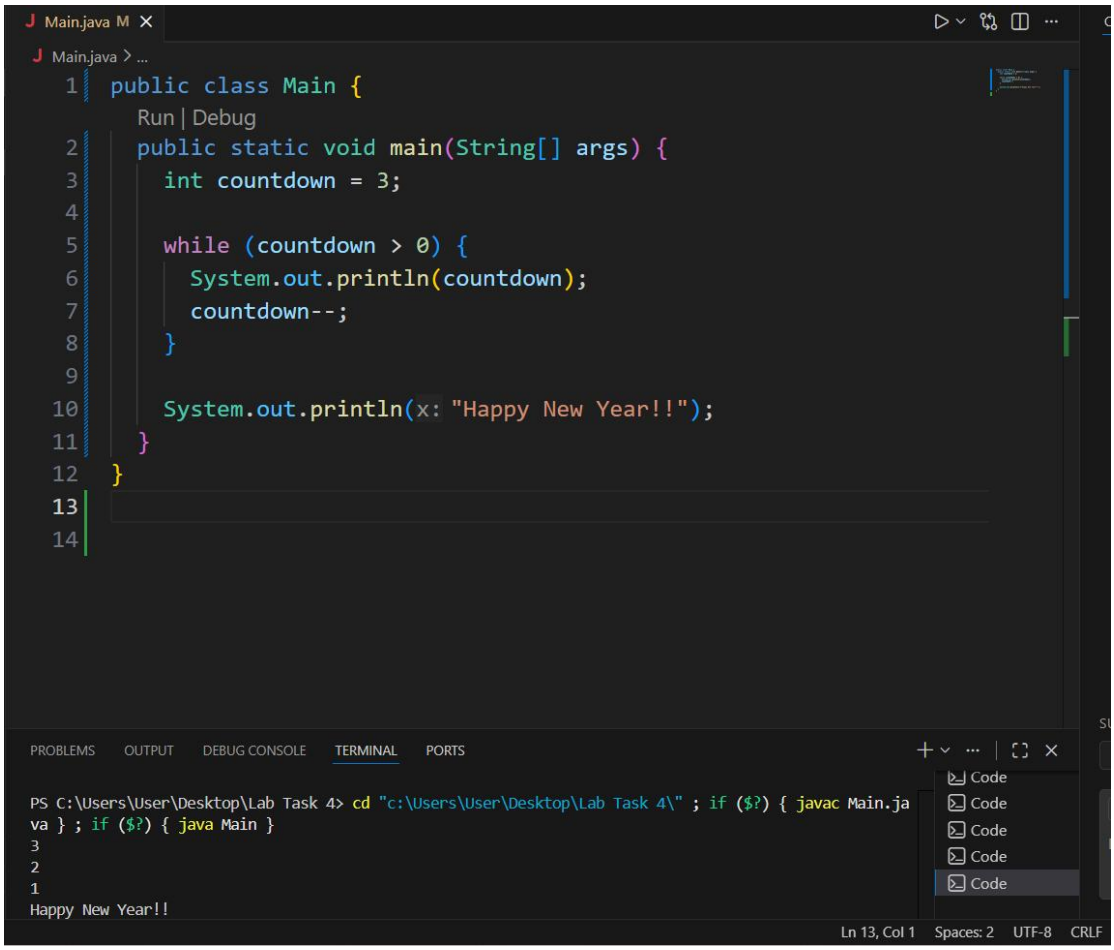


JAVA Real Life While Loop Example

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `countdown` is created and set to `3`.

The program uses a `while` loop that continues as long as `countdown` is greater than `0`. Inside the loop, it prints the current value of `countdown` and then decreases `countdown` by `1` using `countdown--`.

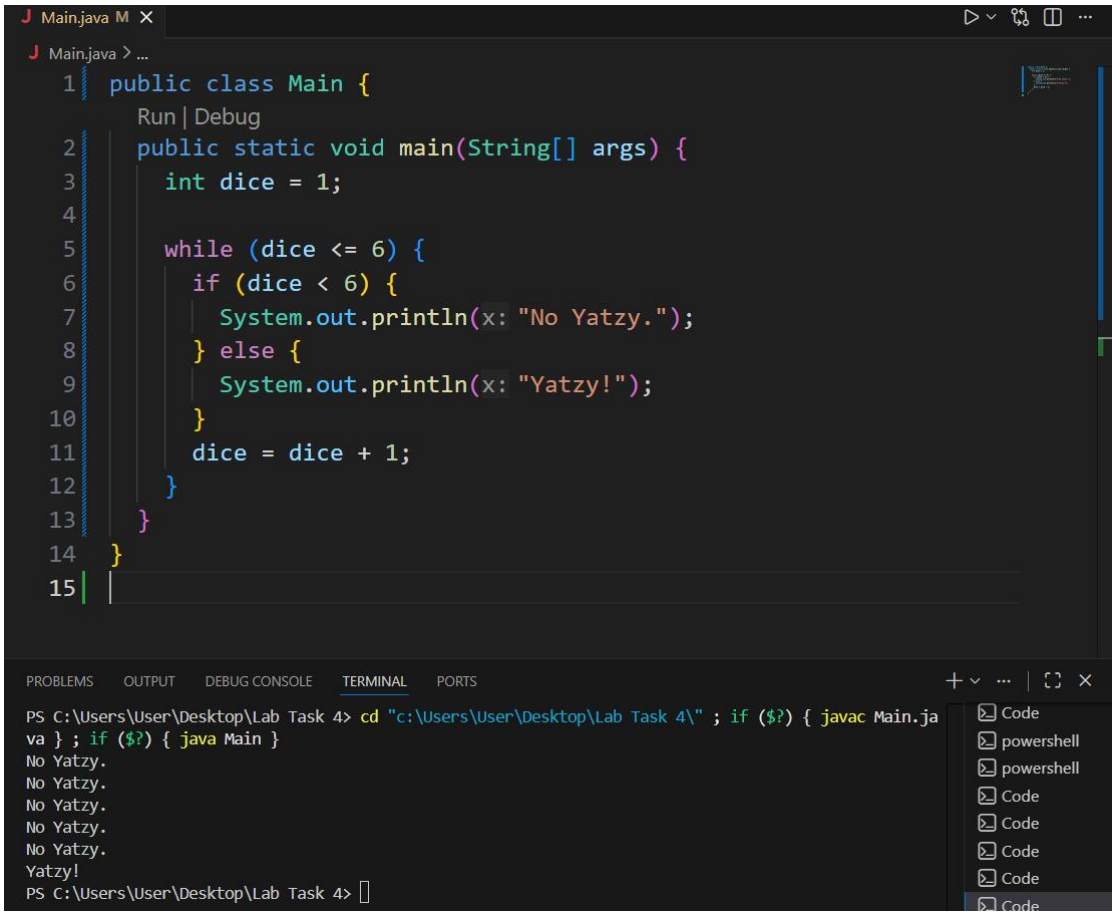
Once `countdown` reaches `0`, the loop stops, and the program prints `"Happy New Year!!"`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `dice` is created and set to `1`.

The program uses a `while` loop that continues as long as `dice` is less than or equal to `6`. Inside the loop, it checks if `dice` is less than `6`. If it is, it prints `"No Yatzy."`. Otherwise, when `dice` equals `6`, it prints `"Yatzy!"`. After each iteration, `dice` is increased by `1`.

This demonstrates how a loop can run multiple times with a condition and use `if-else` inside the loop to handle different cases.

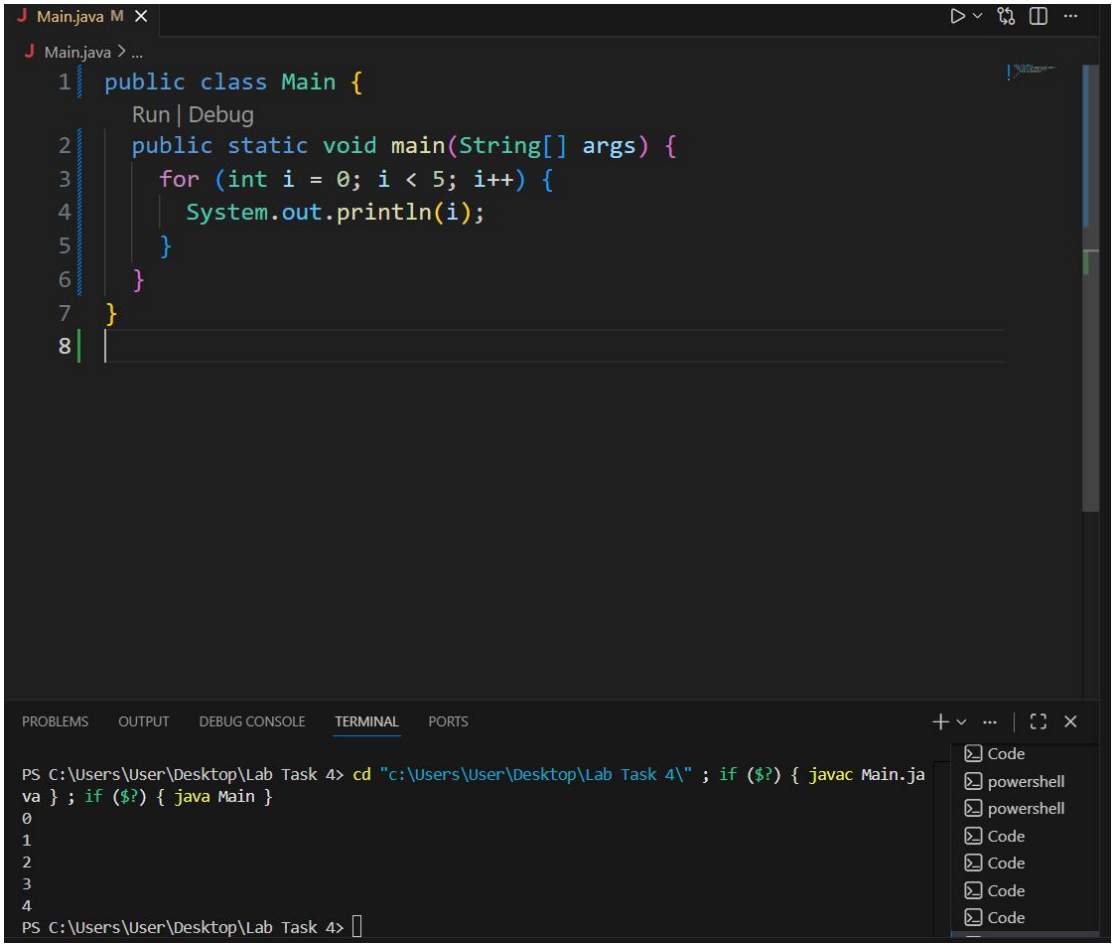


JAVA Print Numbers

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to repeat a block of code a certain number of times.

The loop starts with `i = 0` and continues as long as `i < 5`. After each iteration, `i` is increased by `1` using `i++`. Inside the loop, the current value of `i` is printed.

This shows how a `for` loop can be used to run code repeatedly with a counter.

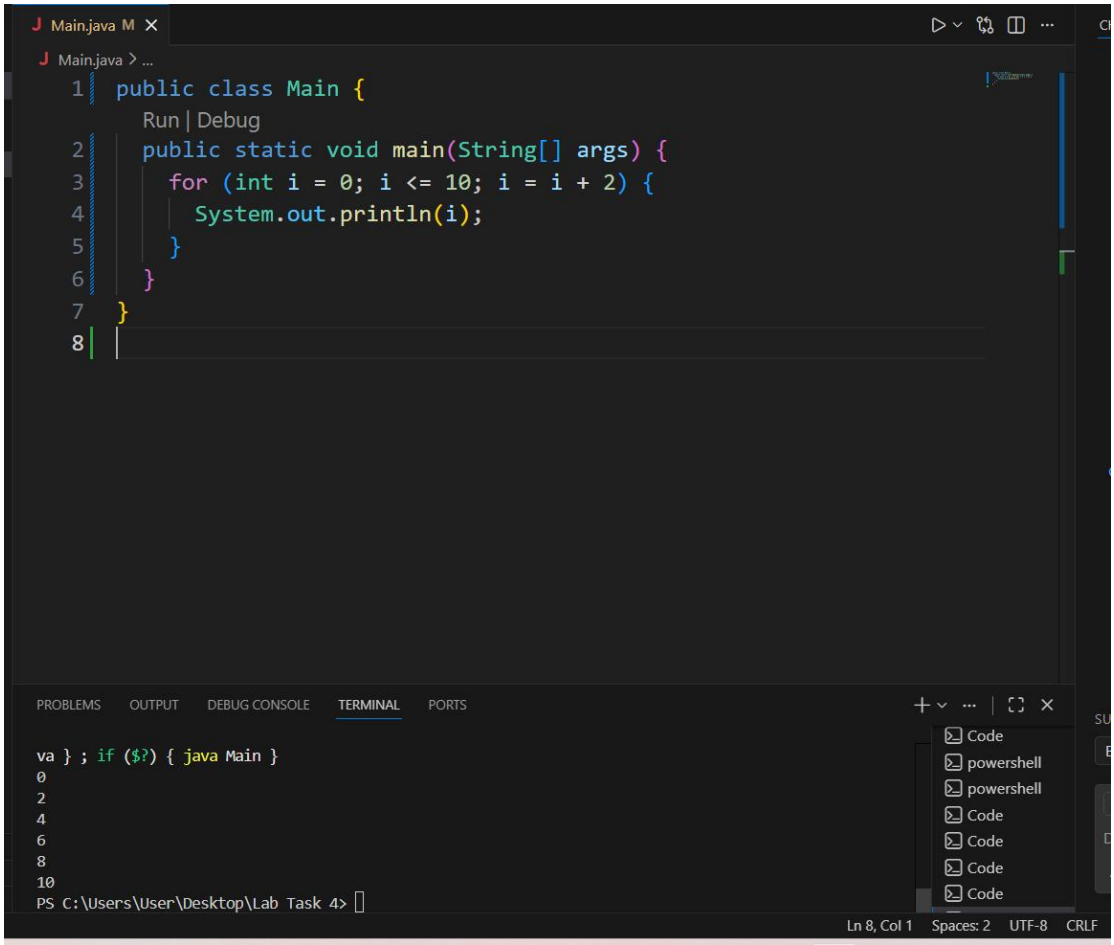


JAVA Print Even Numbers

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to print numbers repeatedly.

The loop starts with `i = 0` and continues as long as `i <= 10`. After each iteration, `i` is increased by `2` using `i = i + 2`. Inside the loop, the current value of `i` is printed.

This shows how a for loop can be used to print numbers with a custom step, in this case, even numbers from 0 to 10.



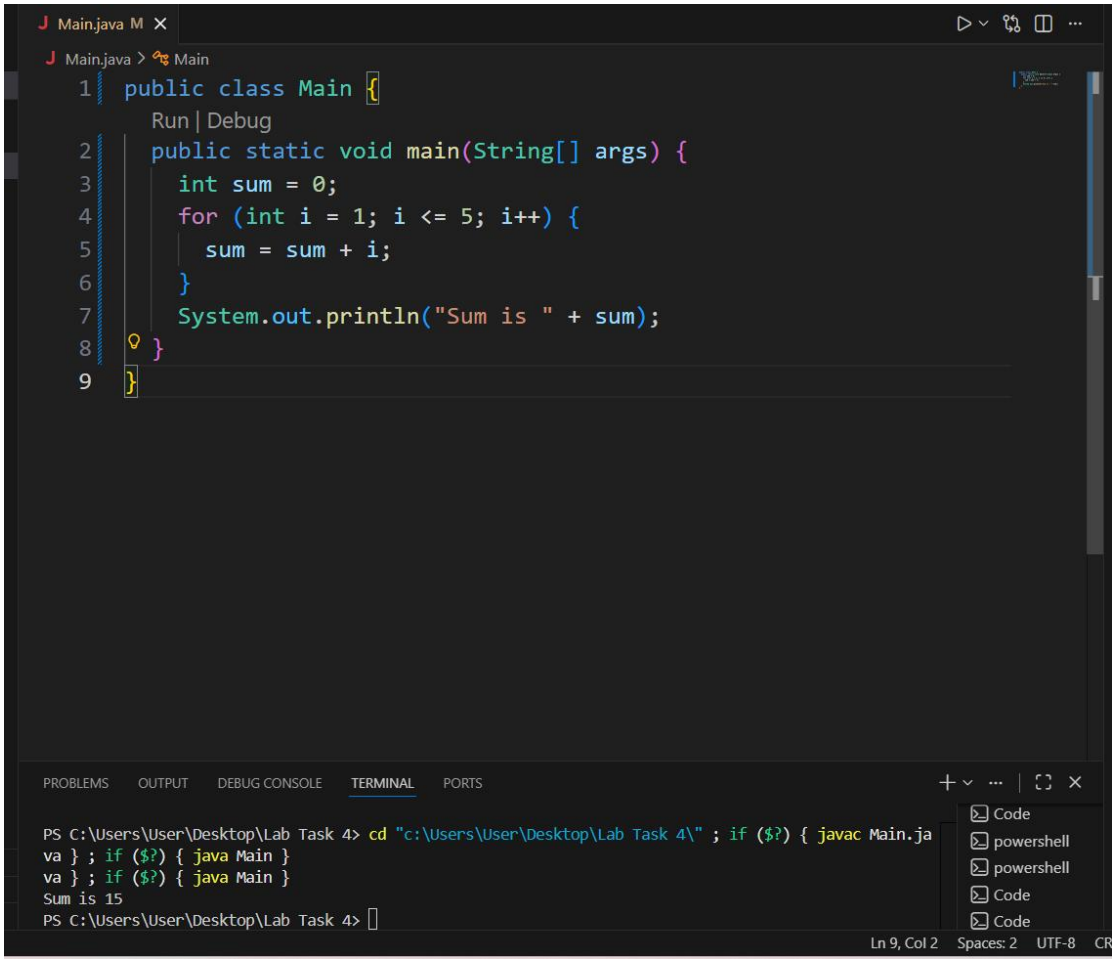
JAVA Sum Of Numbers

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `sum` is created and initialized to `0`.

The program uses a `for` loop that starts with `i = 1` and continues as long as `i <= 5`, increasing `i` by `1` after each iteration. Inside the loop, the current value of `i` is added to `sum`.

After the loop finishes, the program prints the total sum using `System.out.println`.

This demonstrates how a `for` loop can be used to calculate the sum of a series of numbers.

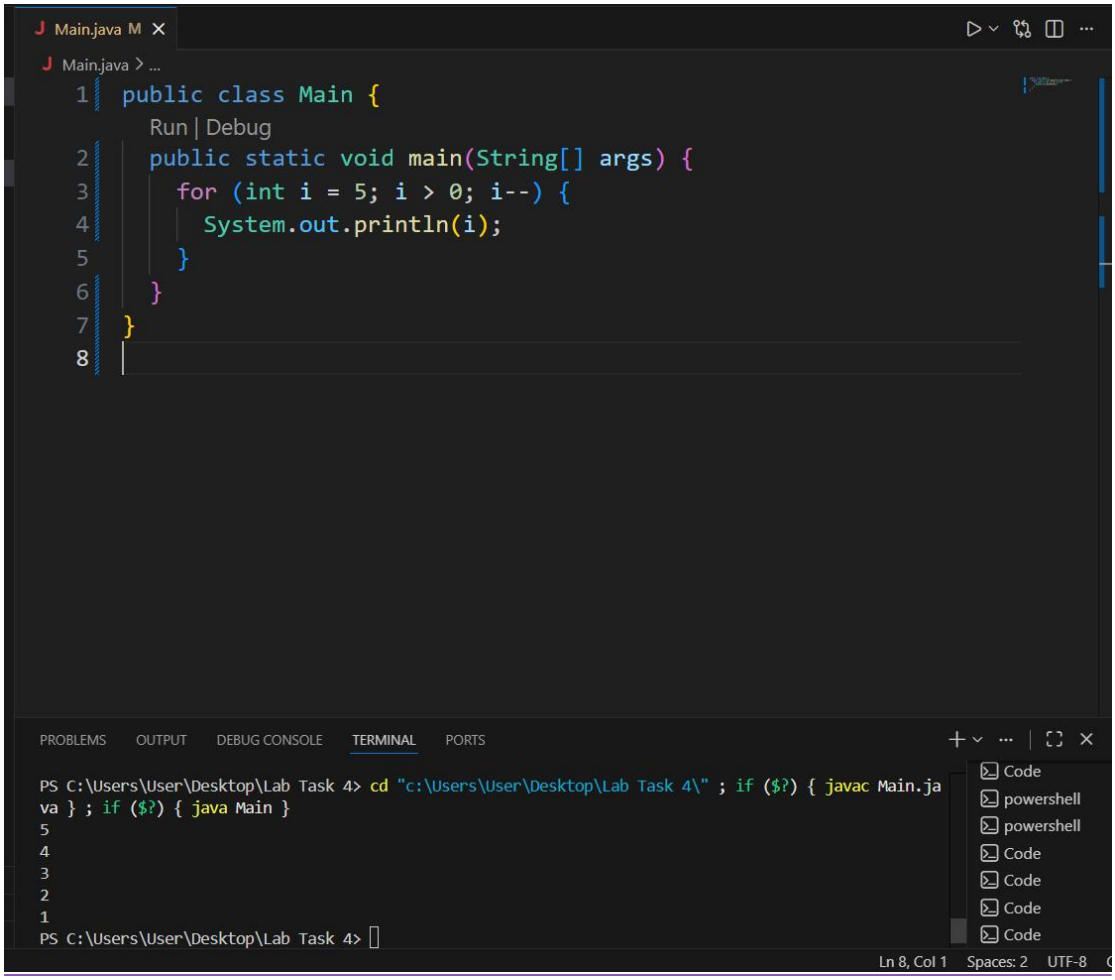


Countdown

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to count down from 5 to 1.

The loop starts with `i = 5` and continues as long as `i > 0`. After each iteration, `i` is decreased by `1` using `i--`. Inside the loop, the current value of `i` is printed.

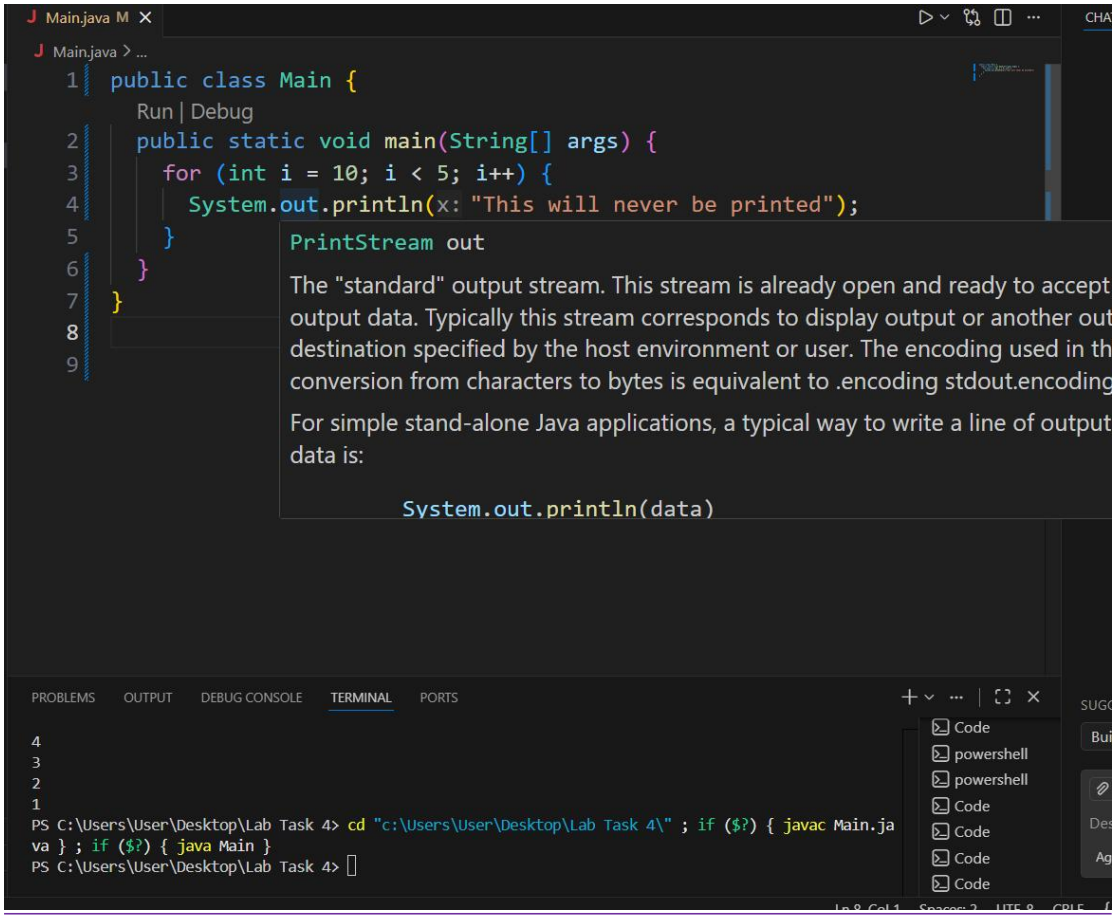
This demonstrates how a for loop can be used to count backward.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is written to start with `i = 10` and continue as long as `i < 5`.

Since `i` is already `10`, which is not less than `5`, the loop condition is false from the start. Therefore, the loop body never executes, and nothing is printed on the screen.

This shows that a `for` loop only runs when its condition is true at the beginning of the loop.



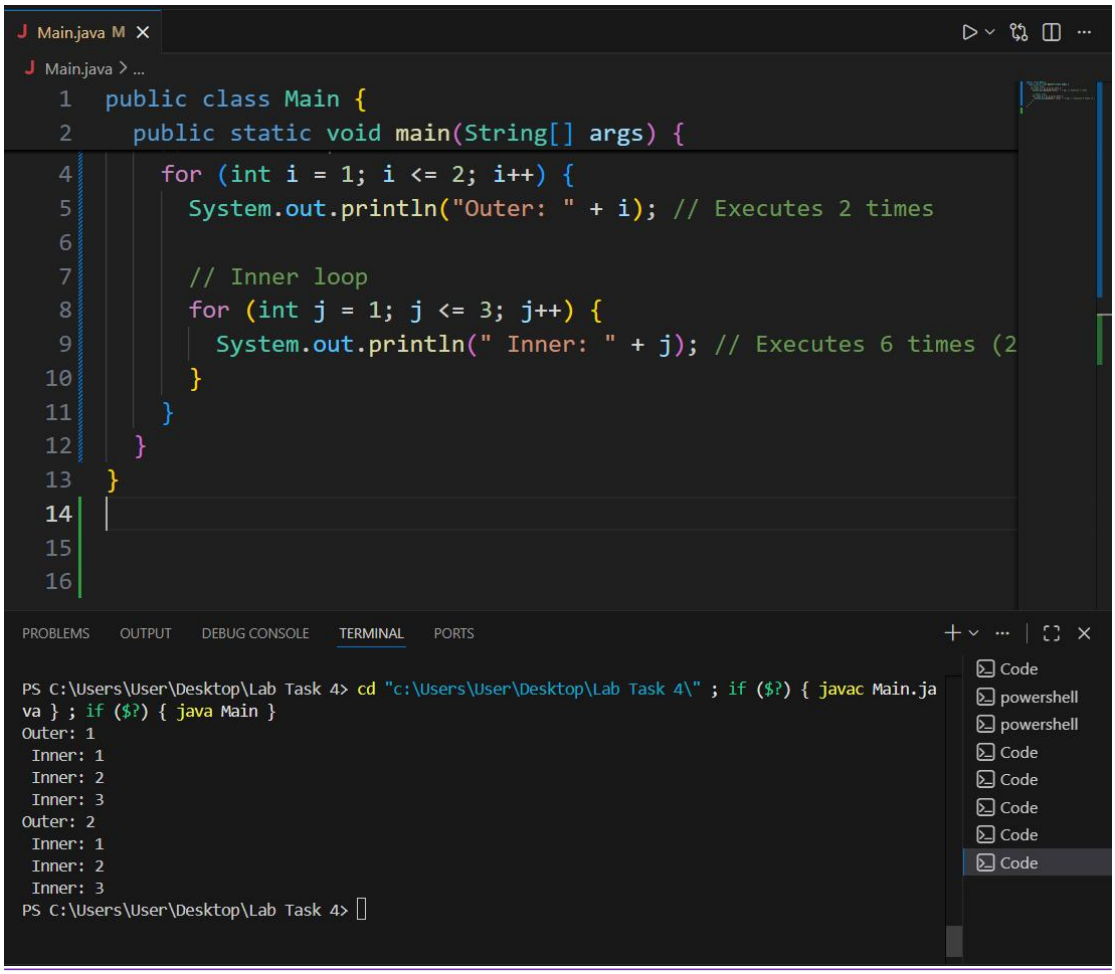
Java Nested Loops

This program defines a class named `Main`, and the `main` method is where the program starts executing. It uses **nested** `for` **loops**, meaning one loop is inside another.

> The **outer loop** runs with `i` from `1` to `2`, so it executes **2 times**.
>
> Inside the outer loop, there is an **inner loop** with `j` from `1` to `3`, which executes **3 times for each iteration of the outer loop**.

During execution, the outer loop prints `"Outer: "` followed by `i`, and the inner loop prints `" Inner: "` followed by `j`.

This shows how nested loops work: for each iteration of the outer loop, the inner loop runs completely.



MulTiplication Table

This program defines a class named `Main`, and the `main` method is where the program starts executing. It uses **nested** `for` **loops** to create a small multiplication table.

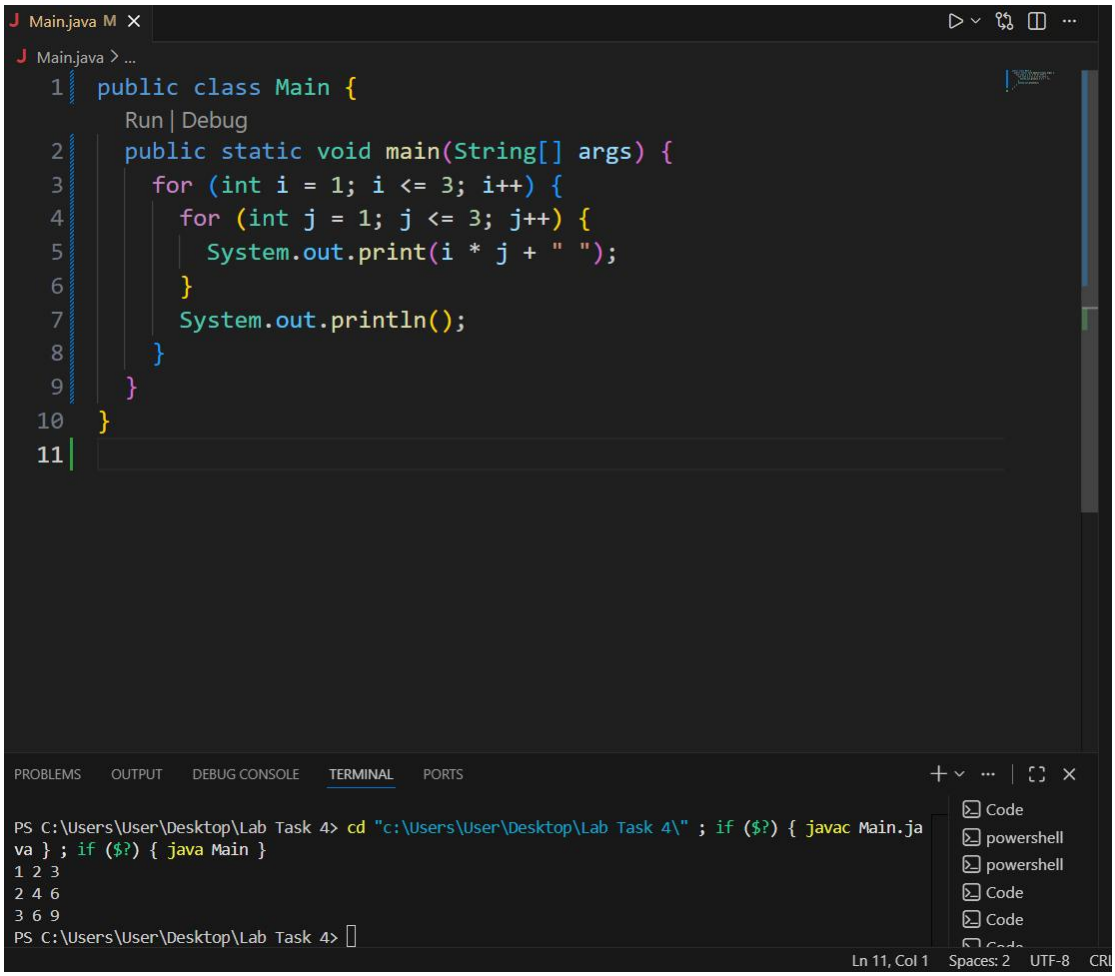
The **outer loop** runs with `i` from `1` to `3`.

The **inner loop** runs with `j` from `1` to `3` for each value of `i`.

Inside the inner loop, the program prints the product `i * j` followed by a space using `System.out.print`.

After the inner loop finishes for one `i`, `System.out.println()` moves to the next line.

This demonstrates how nested loops can be used to generate a grid or table of values.

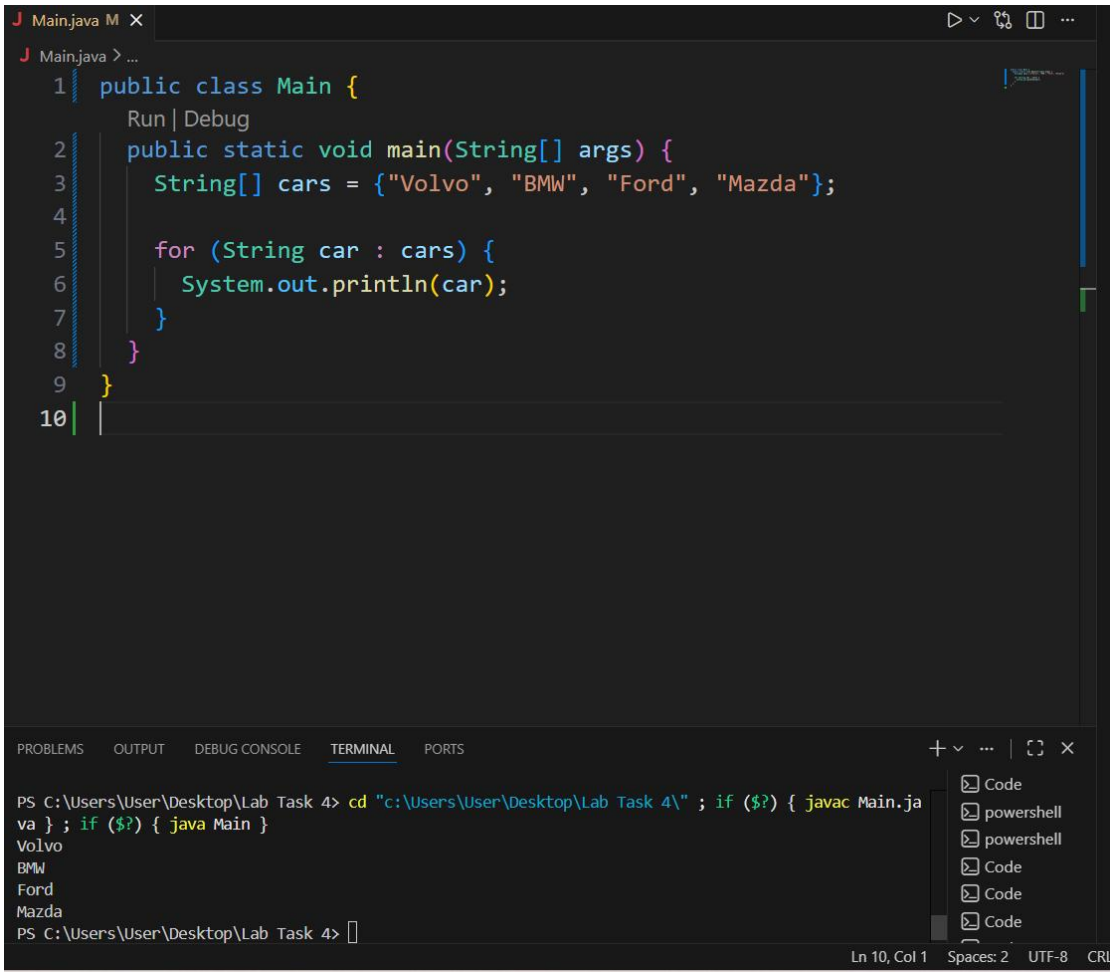


Java For-Each Loop

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an array of strings named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

The program uses a **for-each loop** (`for (String car : cars)`) to go through each element of the array one by one. Inside the loop, it prints the current car name using `System.out.println(car)`.

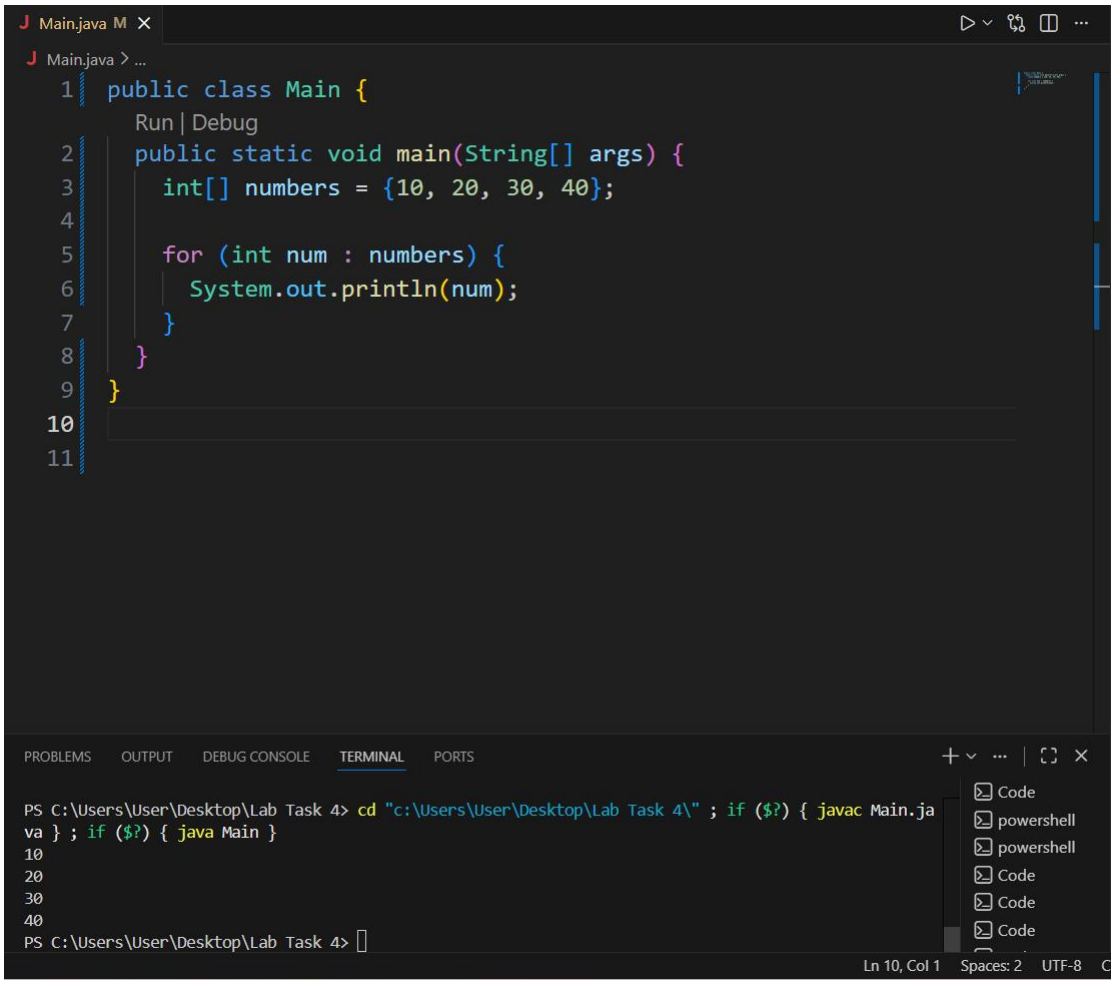
This shows how a for-each loop can be used to easily access every element in an array.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array named `numbers` is created with the values `10`, `20`, `30`, and `40`.

The program uses a **for-each loop** (`for (int num : numbers)`) to go through each element of the array one by one. Inside the loop, it prints the current number using `System.out.println(num)`.

This demonstrates how a for-each loop makes it easy to access and print every element in an array.



Java Real life For Loop Example

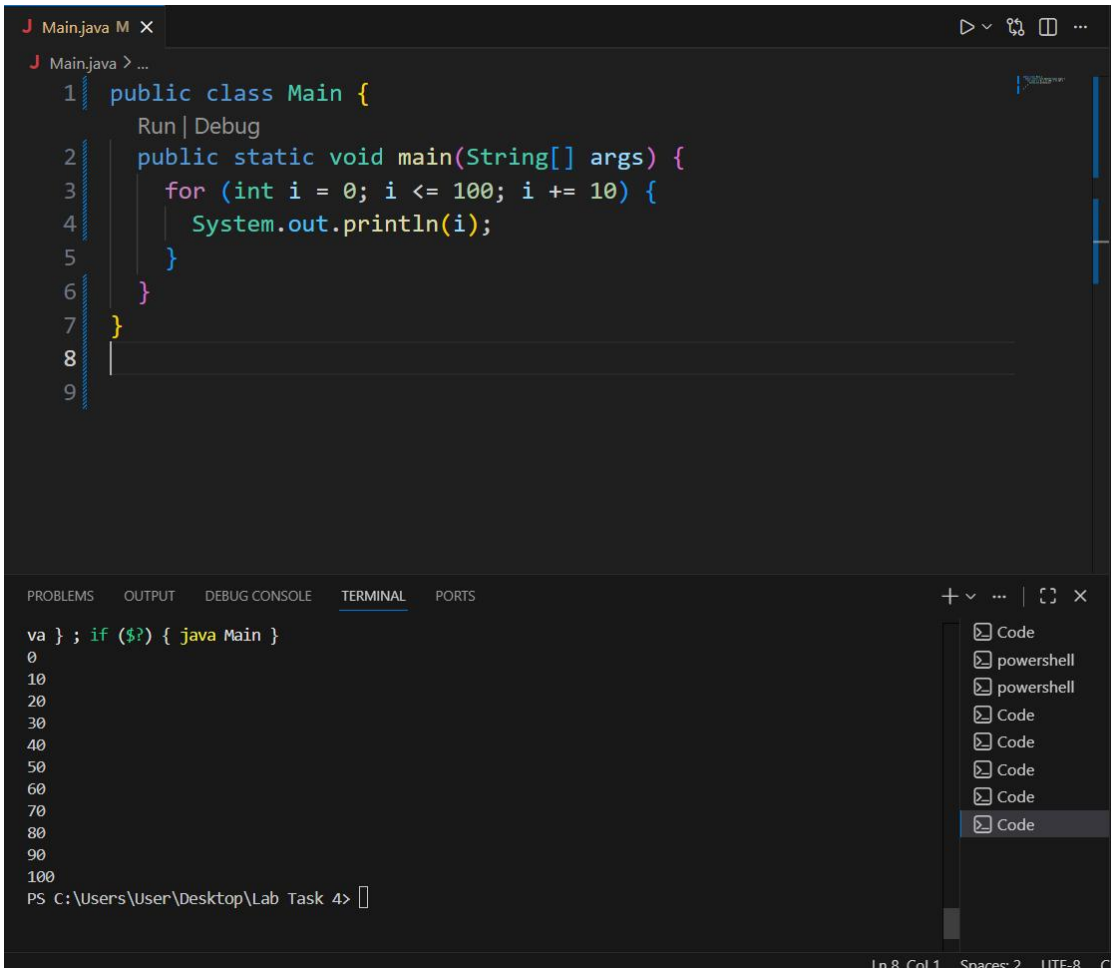
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to print numbers from `0` to `100` in steps of `10`.

The loop starts with `i = 0` and continues as long as `i <= 100`.

After each iteration, `i` is increased by `10` using `i += 10`.

Inside the loop, the current value of `i` is printed.

This demonstrates how a `for` loop can be used to print numbers with a specific step value.



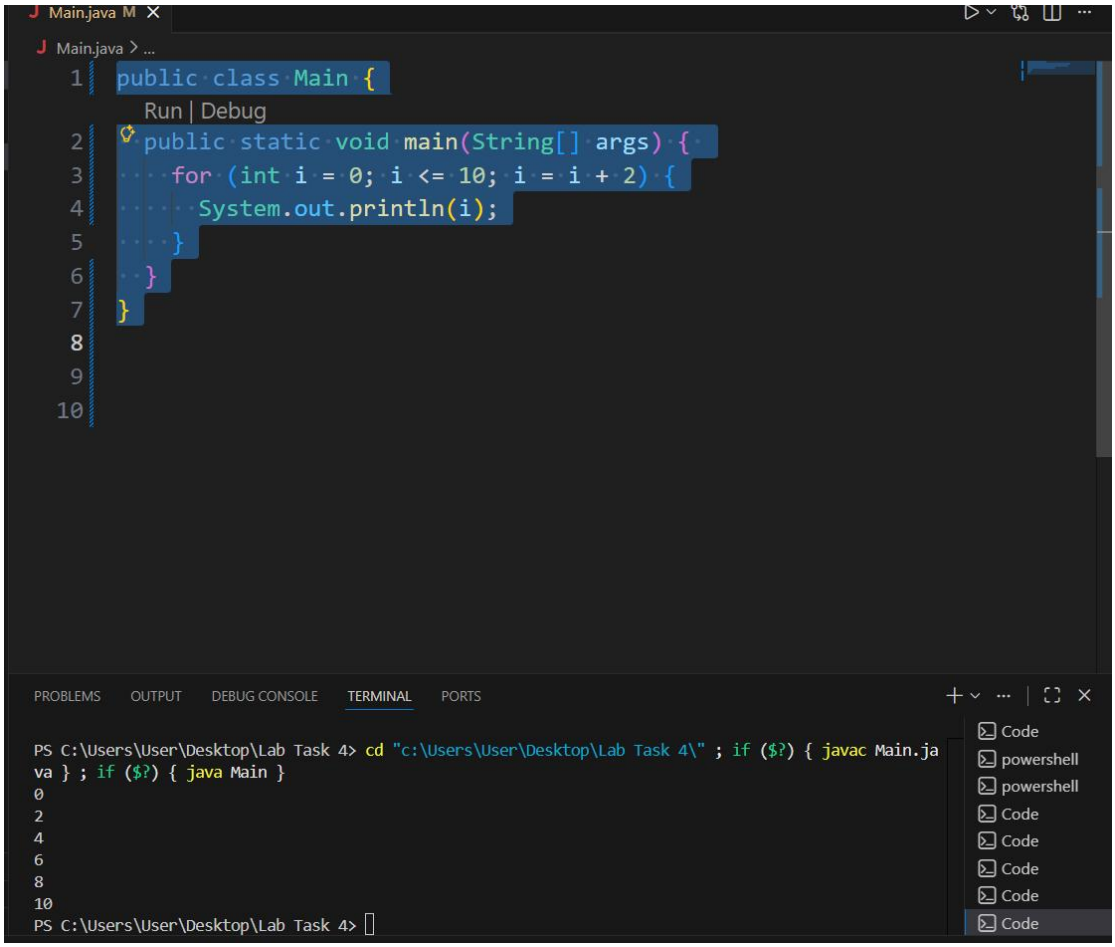
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to print **even numbers from 0 to 10**.

The loop starts with `i = 0` and continues as long as `i <= 10`.

After each iteration, `i` is increased by `2` using `i = i + 2`.

Inside the loop, the current value of `i` is printed using `System.out.println(i)`.

This shows how a for loop can be used to print numbers with a custom step.



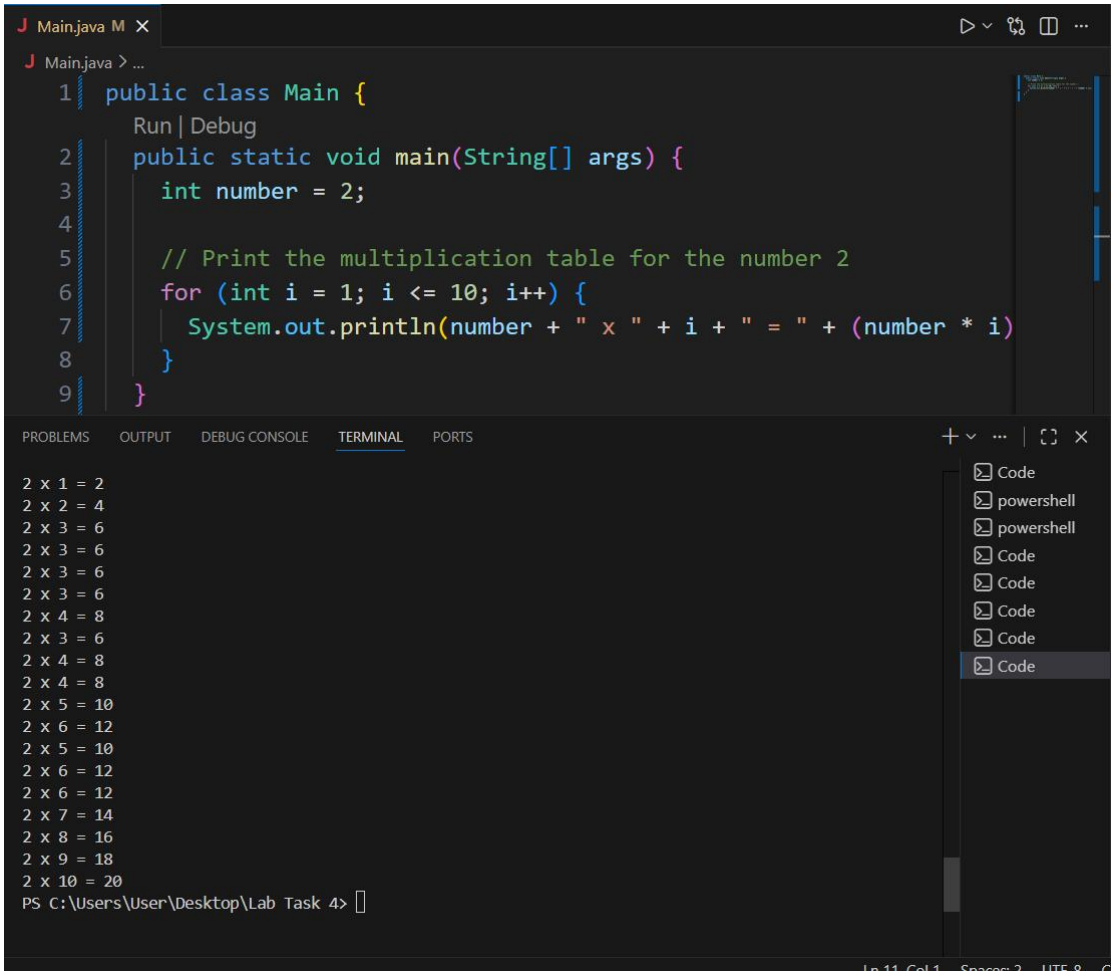
The program uses a `for` loop to print the **multiplication table for 2**.

The loop starts with `i = 1` and continues as long as `i <= 10`.

Inside the loop, it prints the product of `number` and `i` in the format `"2 x i = result"` using `System.out.println(number + " x " + i + " = " + (number * i))`.

After each iteration, `i` increases by `1`.

This demonstrates how a for loop can be used to generate a multiplication table.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop is used to print seat numbers from `1` to `5`.

The loop starts with `seat = 1` and continues as long as `seat <= 5`.

After each iteration, `seat` increases by `1` using `seat++`.

Inside the loop, the current seat number is printed using `System.out.println("Seat number: " + seat)`.

This demonstrates how a `for` loop can be used to repeat a task a fixed number of times.

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer variable `n` is created and set to `5`. Another variable `fact` is created to store the factorial and initialized to `1`.

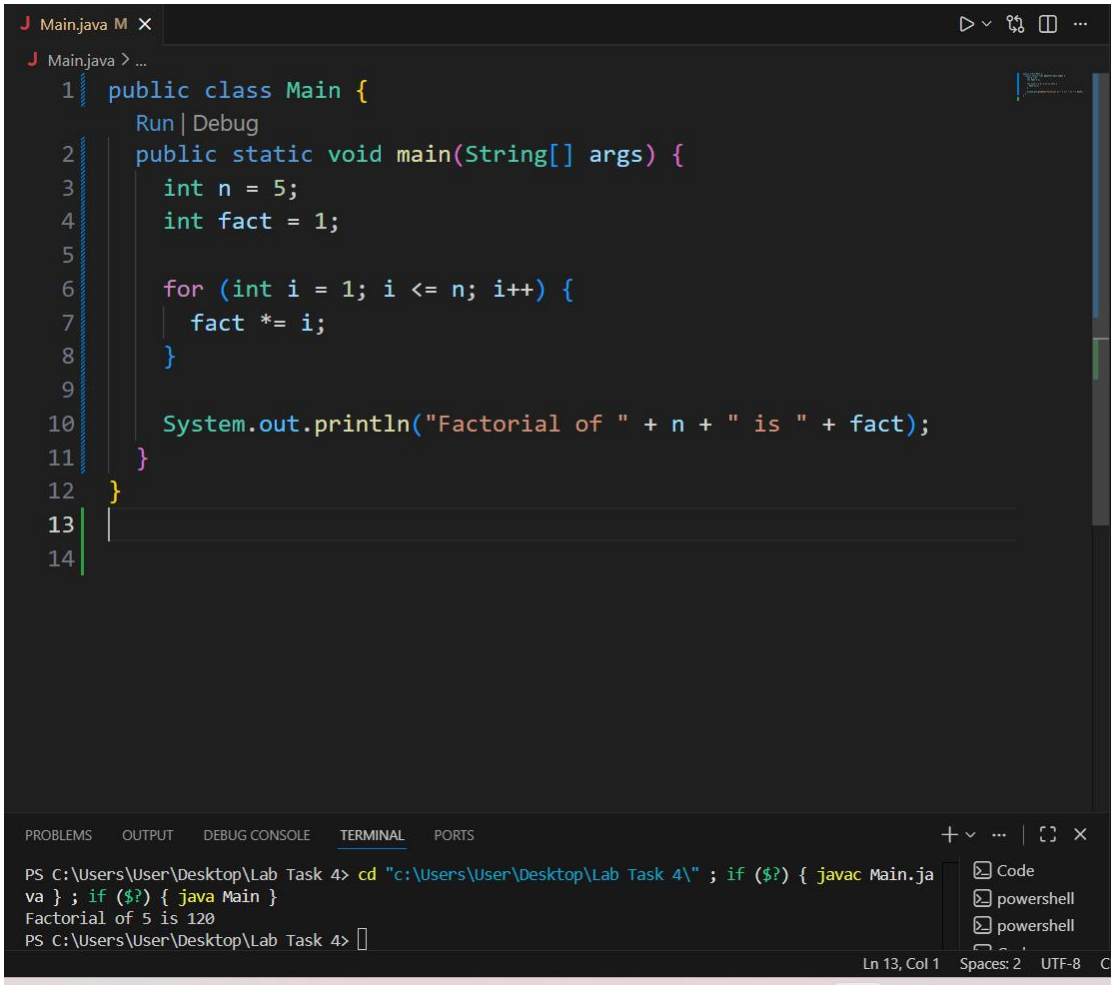
The program uses a `for` loop to calculate the factorial of `n`.

The loop starts with `i = 1` and continues as long as `i <= n`.

In each iteration, `fact` is multiplied by `i` using `fact *= i`.

After the loop finishes, the program prints the factorial of `5`.

This demonstrates how a for loop can be used to calculate the factorial of a number.



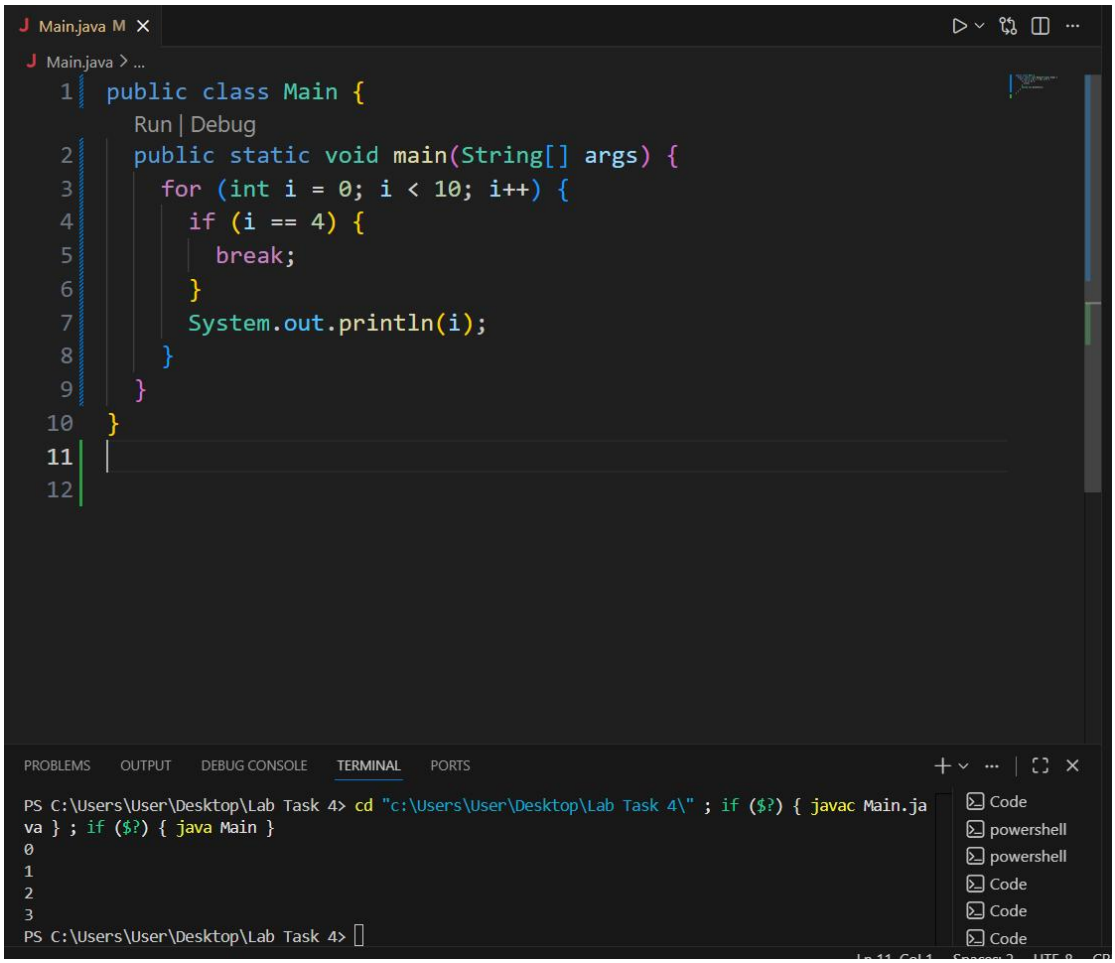
Java Break And Continue

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop runs with `i` starting from `0` and continues as long as `i < 10`.

Inside the loop, there is an `if` statement that checks if `i == 4`. If this condition is true, the `break` statement stops the loop immediately. Otherwise, it prints the current value of `i`.

Since the loop stops when `i` becomes `4`

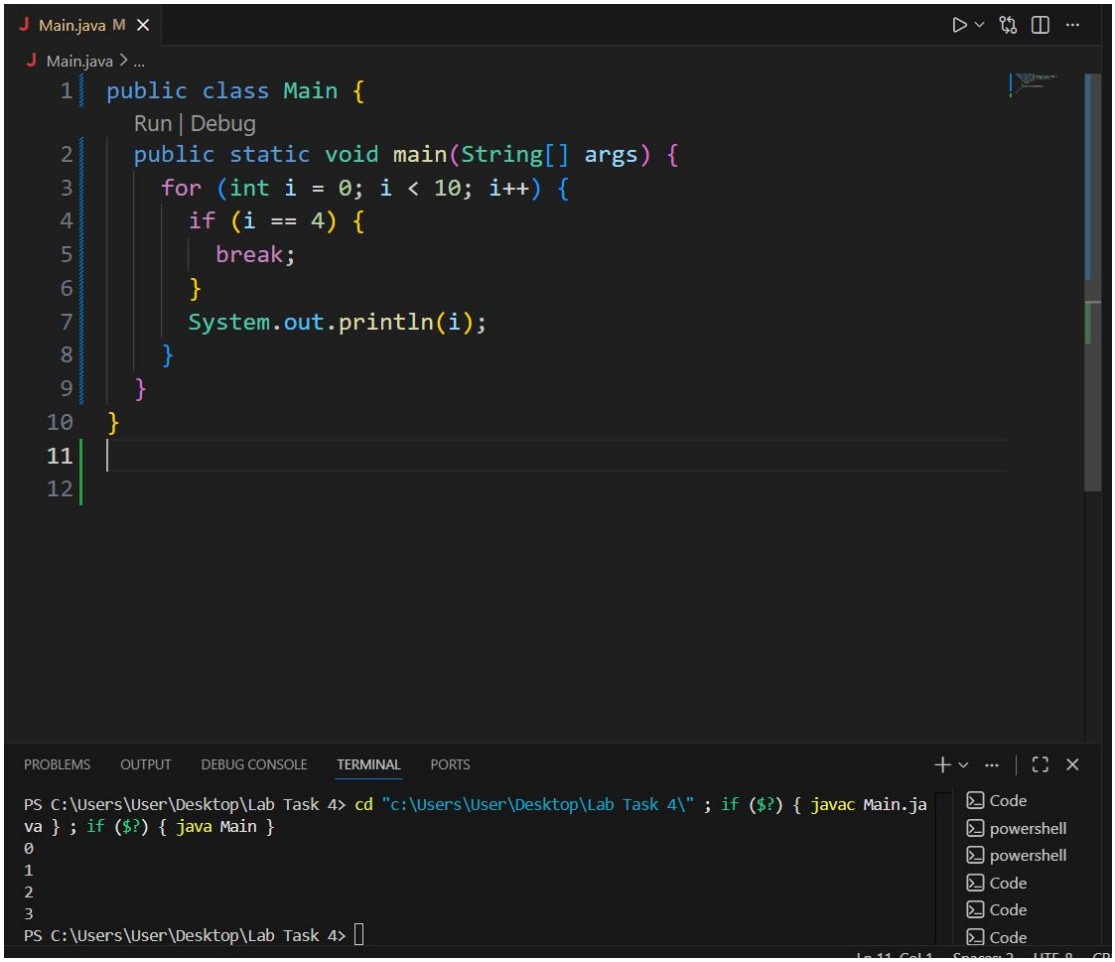
This shows how the break statement can be used to exit a loop early.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop runs with `i` starting from `0` and continues as long as `i < 10`.

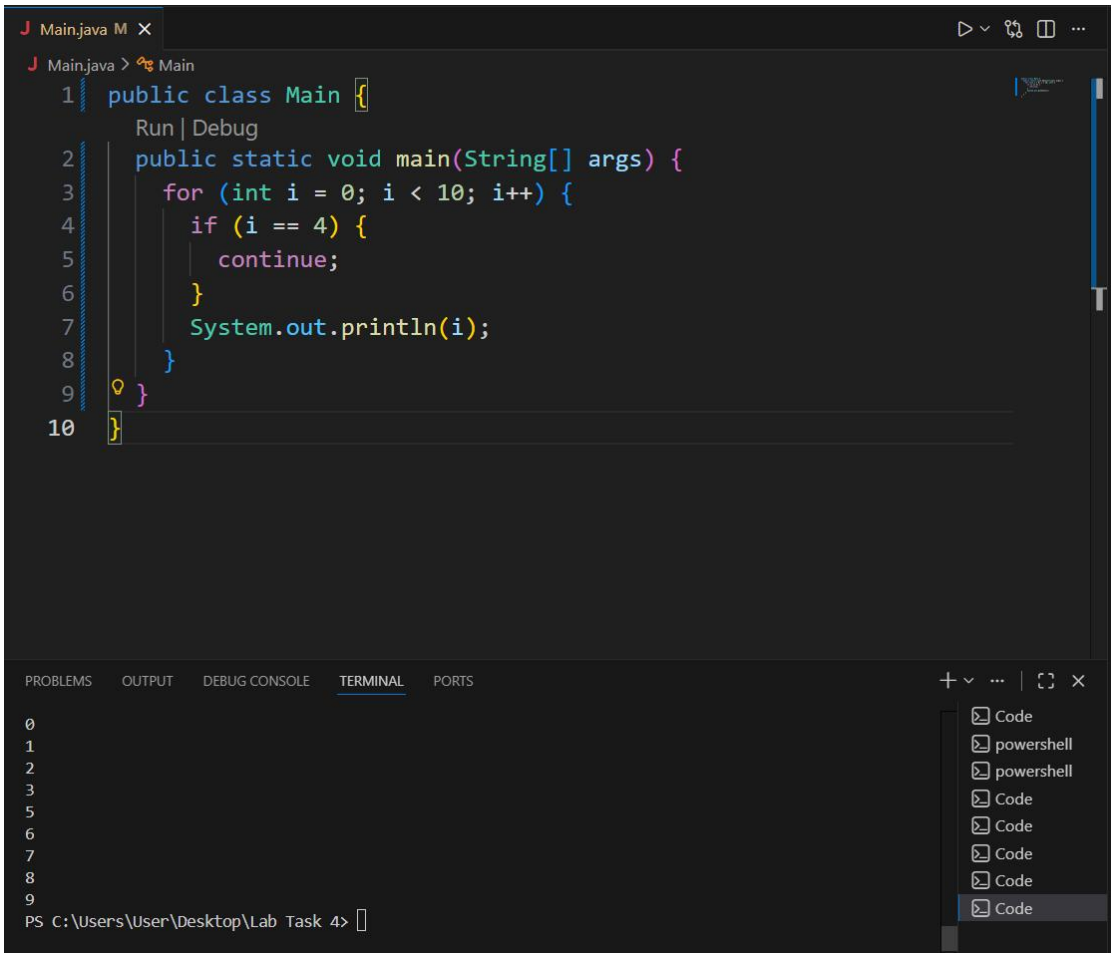
Inside the loop, there is an `if` statement that checks if `i == 4`. If this condition is true, the `continue` statement **skips the current iteration** and moves to the next value of `i`. Otherwise, it prints the current value of `i`.

This shows how the `continue` statement can be used to skip a specific iteration in a loop without stopping the entire loop.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop runs with `i` starting from `0` and continues as long as `i < 10`.

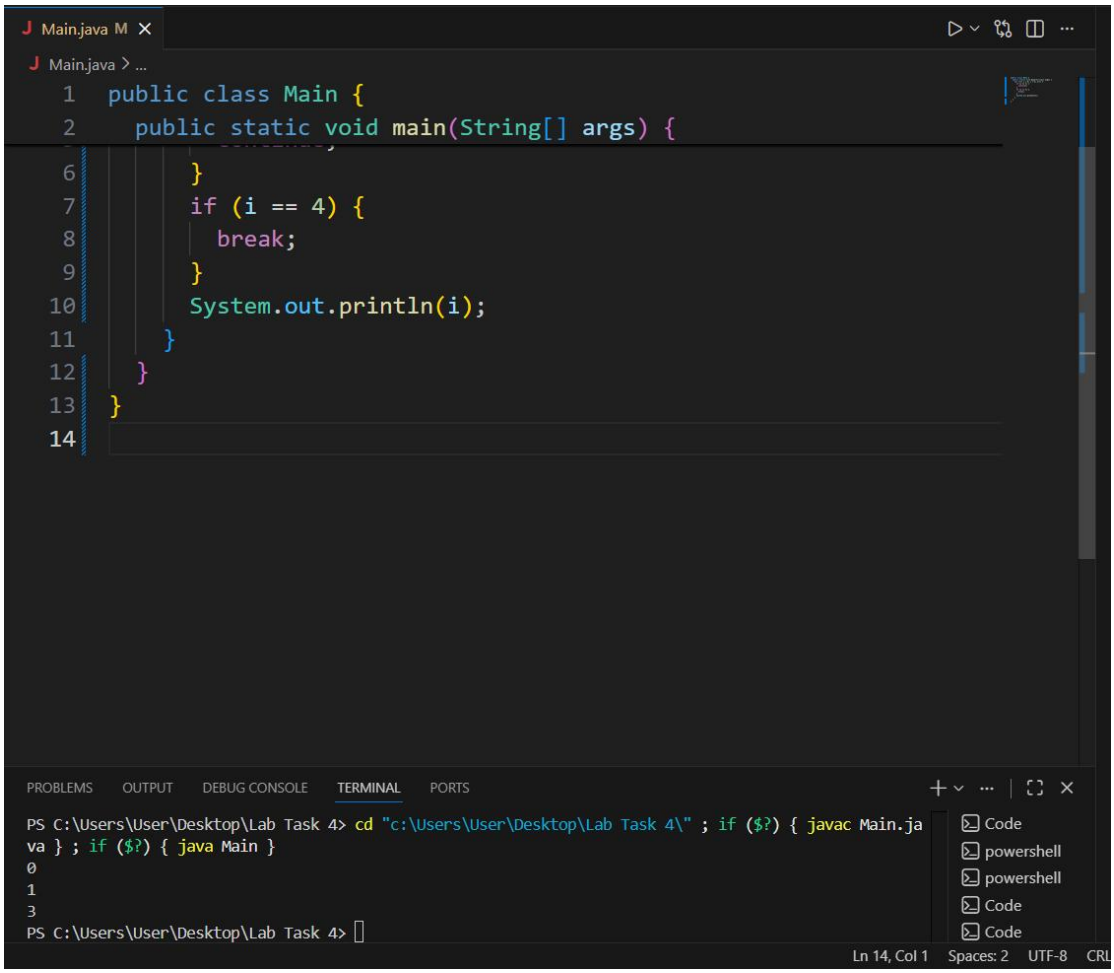
Inside the loop, there is an `if` statement that checks if `i == 4`. If this condition is true, the `continue` statement **skips that iteration** and moves to the next value of `i`. For all other values, the program prints the current value of `i`.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `for` loop runs with `i` starting from `0` and continues as long as `i < 6`.

> The first `if` statement checks if `i == 2`. If it is true, the `continue` statement **skips that iteration**, so `2` is not printed.
>
> The second `if` statement checks if `i == 4`. If it is true, the `break` statement **stops the loop completely**, so the loop ends.
>
> For all other values, the program prints the current value of `i`.

This shows how `continue` can skip a specific iteration and `break` can stop the loop entirely.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `while` loop runs as long as `i < 10`.

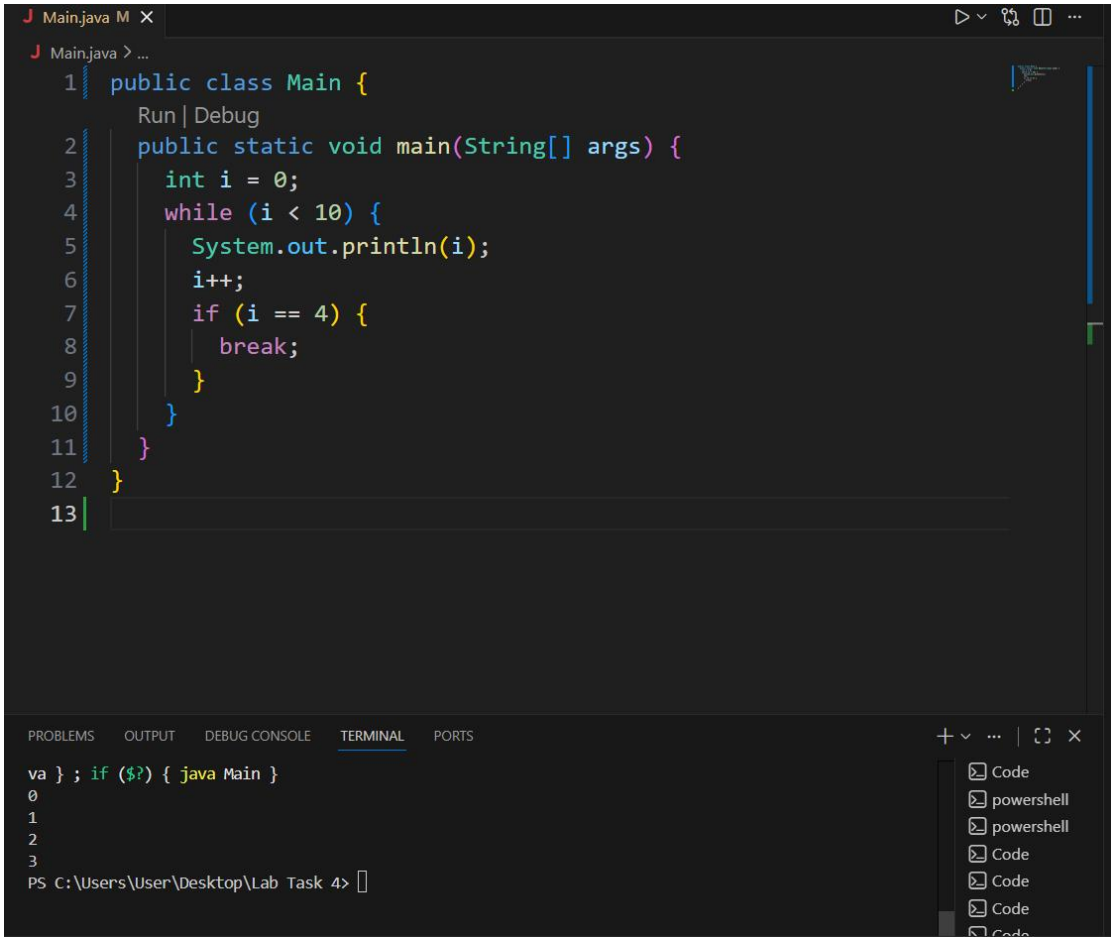
The loop starts with `i = 0`.

Inside the loop, it first prints the current value of `i` using `System.out.println(i)`.

Then `i` is increased by `1` using `i++`.

After that, there is an `if` statement that checks if `i == 4`. If it is true, the `break` statement **stops the loop immediately**.

This demonstrates how a `while` loop works and how the `break` statement can be used to exit the loop early.



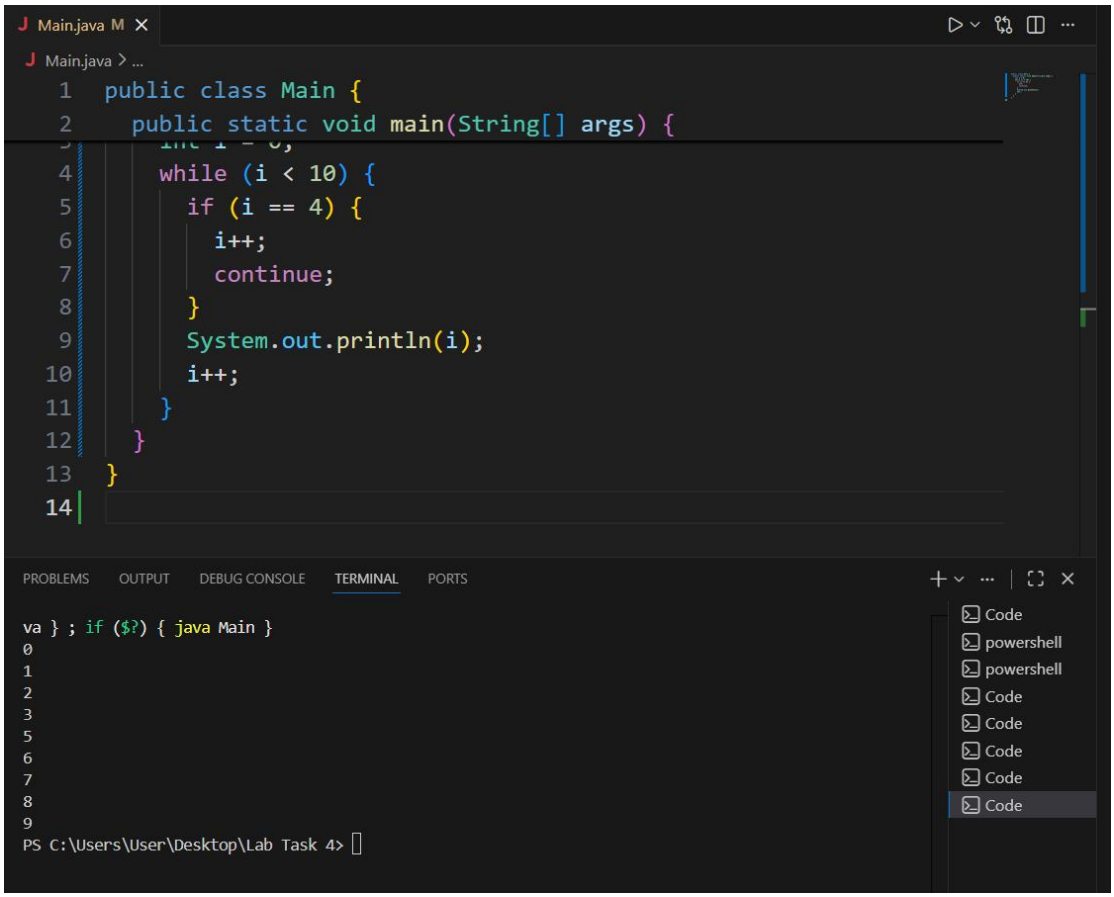
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a `while` loop runs as long as `i < 10`.

The loop starts with `i = 0`.

Inside the loop, there is an `if` statement that checks if `i == 4`. If this is true, `i` is increased by `1` and the `continue` statement **skips the rest of the loop body for that iteration**, so `4` is not printed.

For all other values, the current value of `i` is printed, and then `i` is increased by `1`.

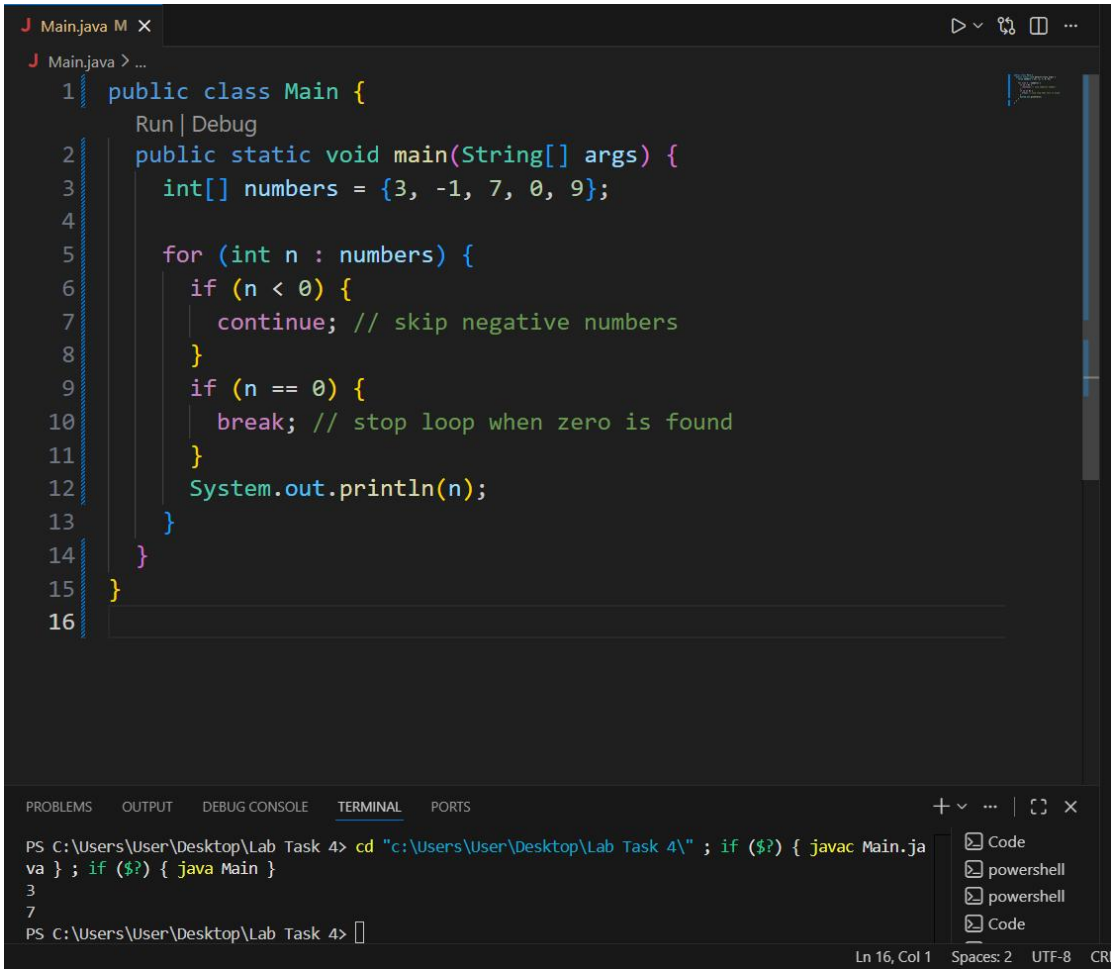
This demonstrates how `continue` can skip a specific iteration in a `while` loop while allowing the loop to continue.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array `numbers` is created with the values `{3, -1, 7, 0, 9}`.

The program uses a **for-each loop** to go through each number in the array:

> If the number is **negative** (`n < 0`), the `continue` statement **skips that number** and moves to the next iteration.
>
> If the number is **zero** (`n == 0`), the `break` statement **stops the loop completely**.
>
> Otherwise, the number is printed using `System.out.println(n)`.
>
> This shows how continue can skip certain values and break can stop a loop early.

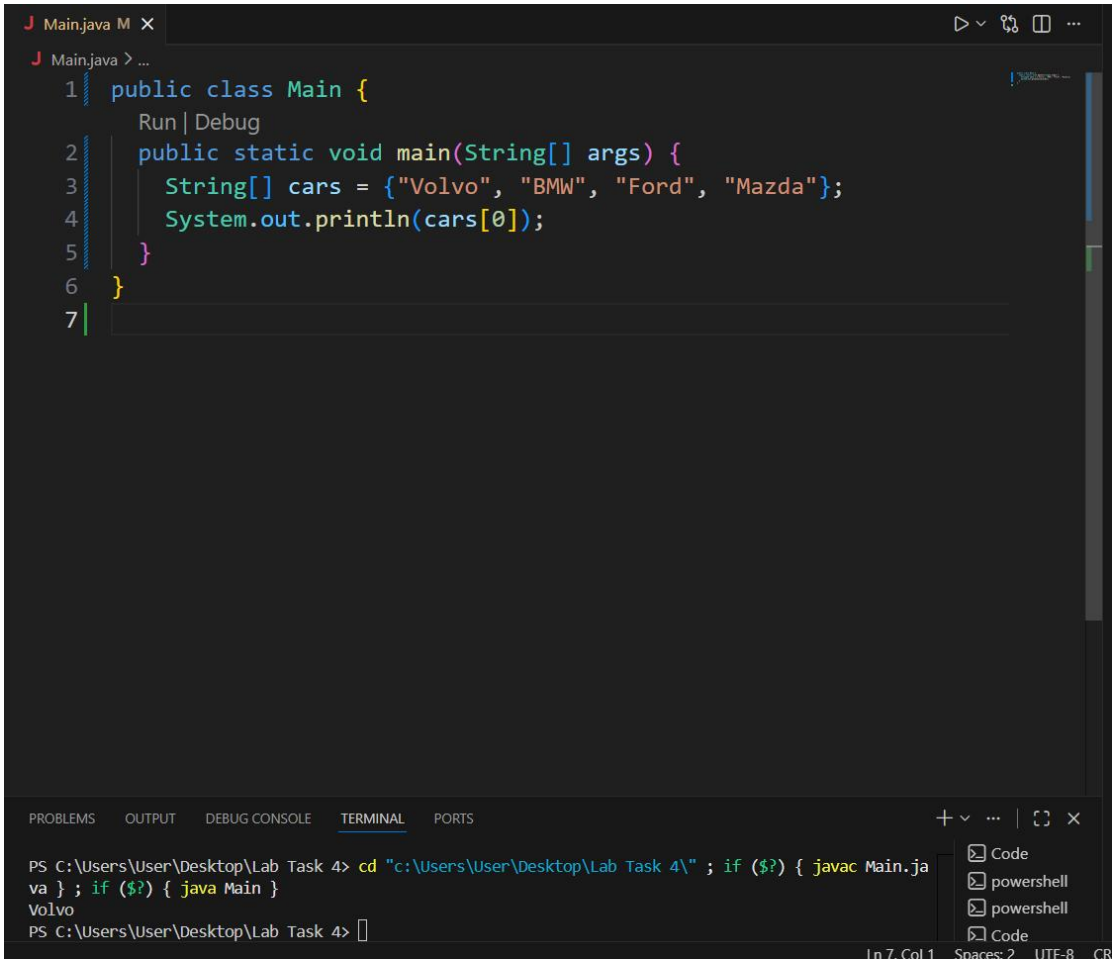


JAVA Arrays

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

The program then prints the **first element** of the array using `cars[0]`. In arrays, indexing starts at `0`, so `cars[0]` refers to `"Volvo"`.

This demonstrates how to access a specific element from an array using its index.

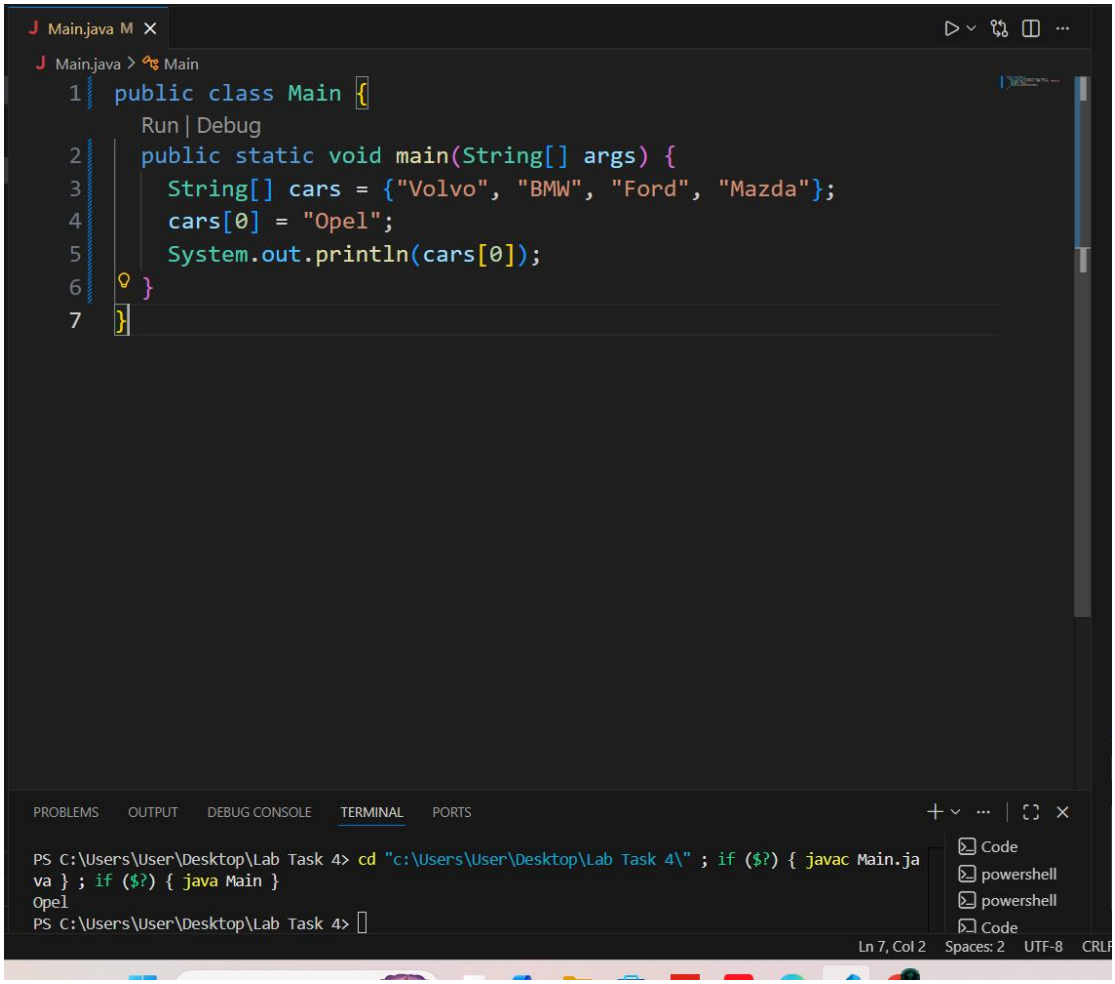


This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

Then, the first element of the array (`cars[0]`) is **changed** from `"Volvo"` to `"Opel"` using `cars[0] = "Opel"`.

Finally, the program prints the first element of the array using `System.out.println(cars[0])`.

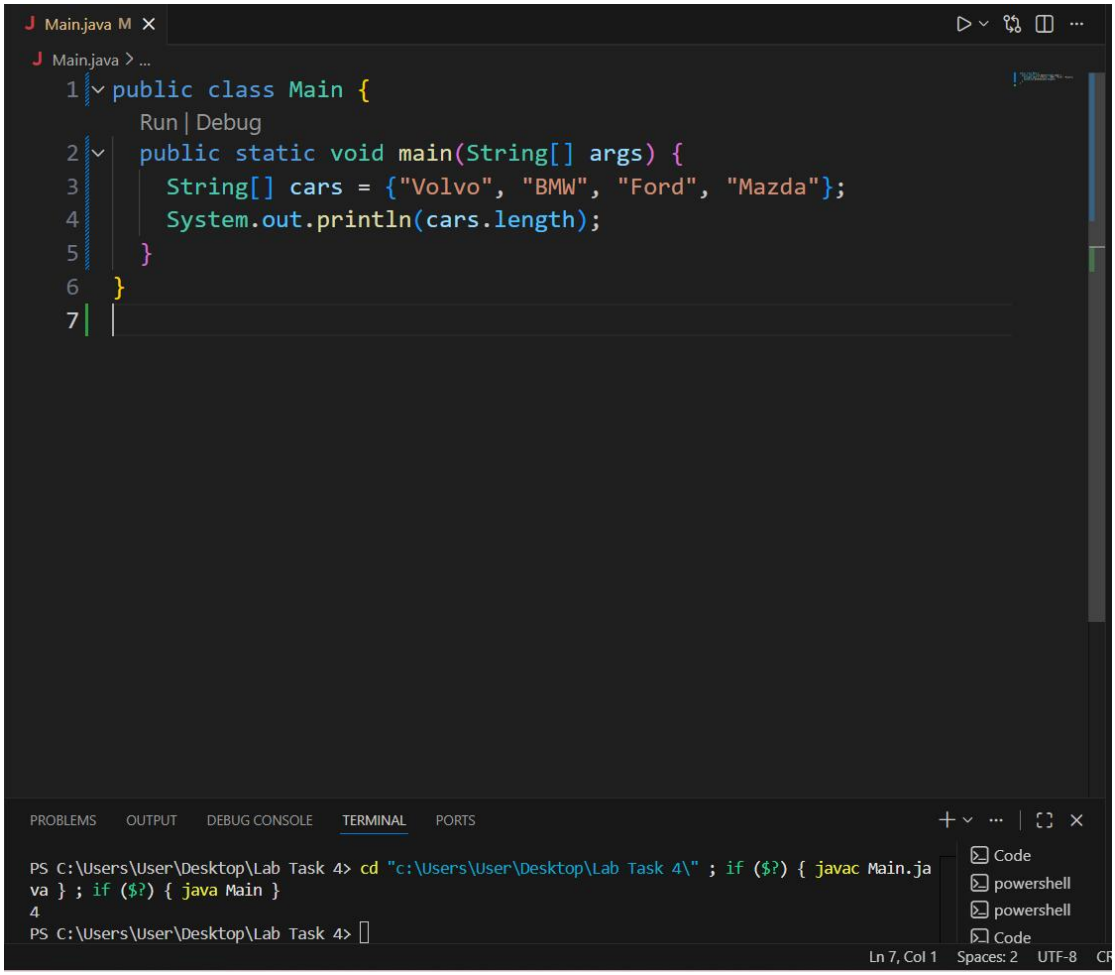
This shows how to **modify a specific element** in an array.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

The program then prints the **length of the array** using `cars.length`. The `length` property tells us how many elements are in the array.

This demonstrates how to find the **number of elements in an array**.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with a **fixed size of 4** using `new String[4]`.

Each element of the array is then assigned a value:

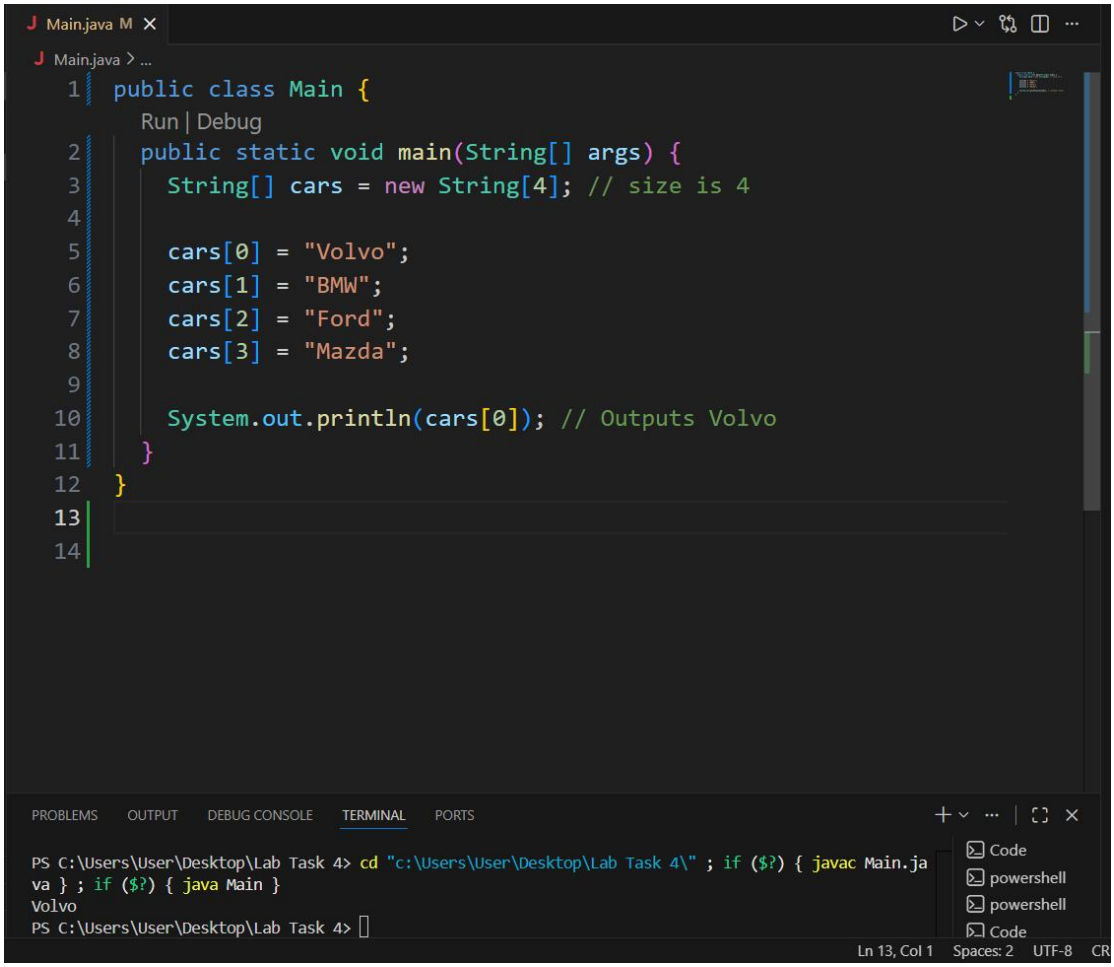
```
cars[0] = "Volvo"

cars[1] = "BMW"

cars[2] = "Ford"

cars[3] = "Mazda"
```

Finally, the program prints the first element of the array using `System.out.println(cars[0])`.



Java Loop Through an array

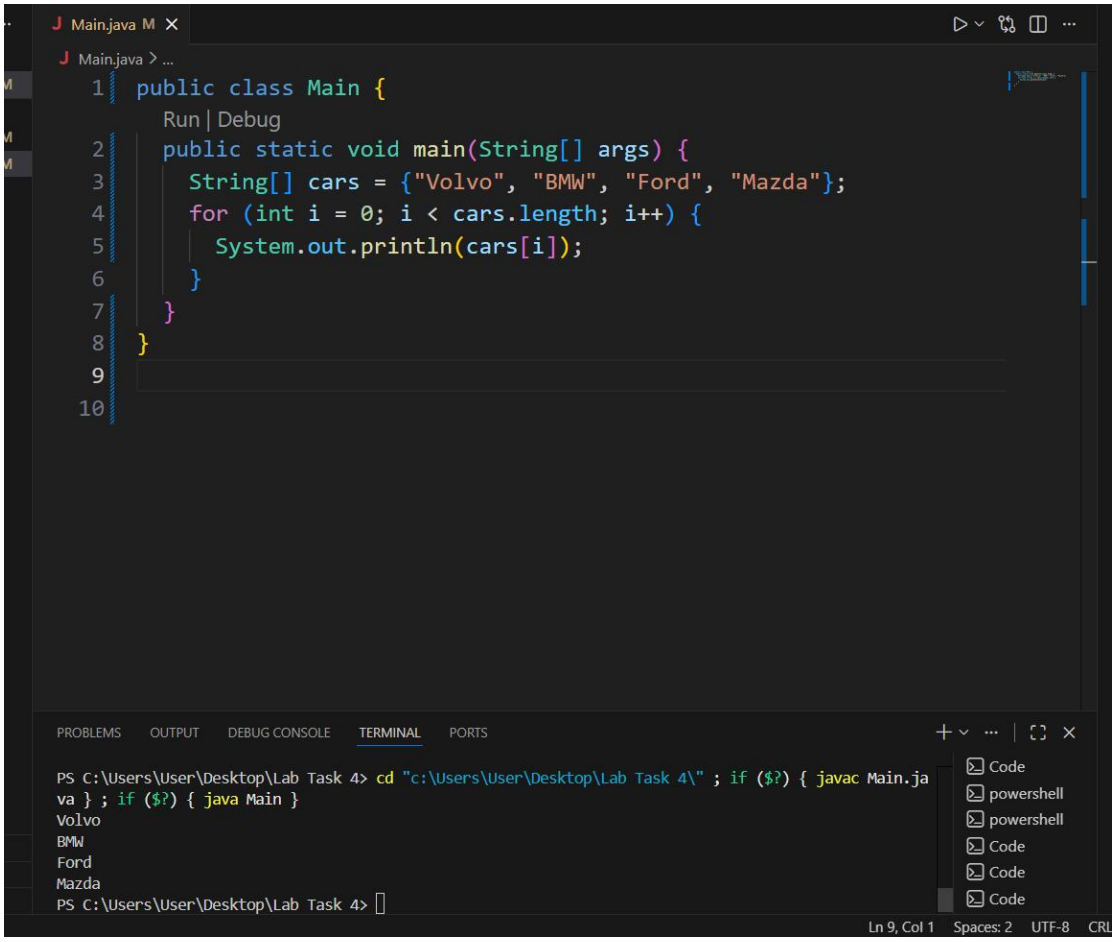
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

The program then uses a **for loop** to go through each element of the array:

The loop starts with `i = 0` and continues as long as `i < cars.length`.

Inside the loop, `cars[i]` prints the element at index `i`.

This demonstrates how a for loop can be used to access and print all elements of an array.



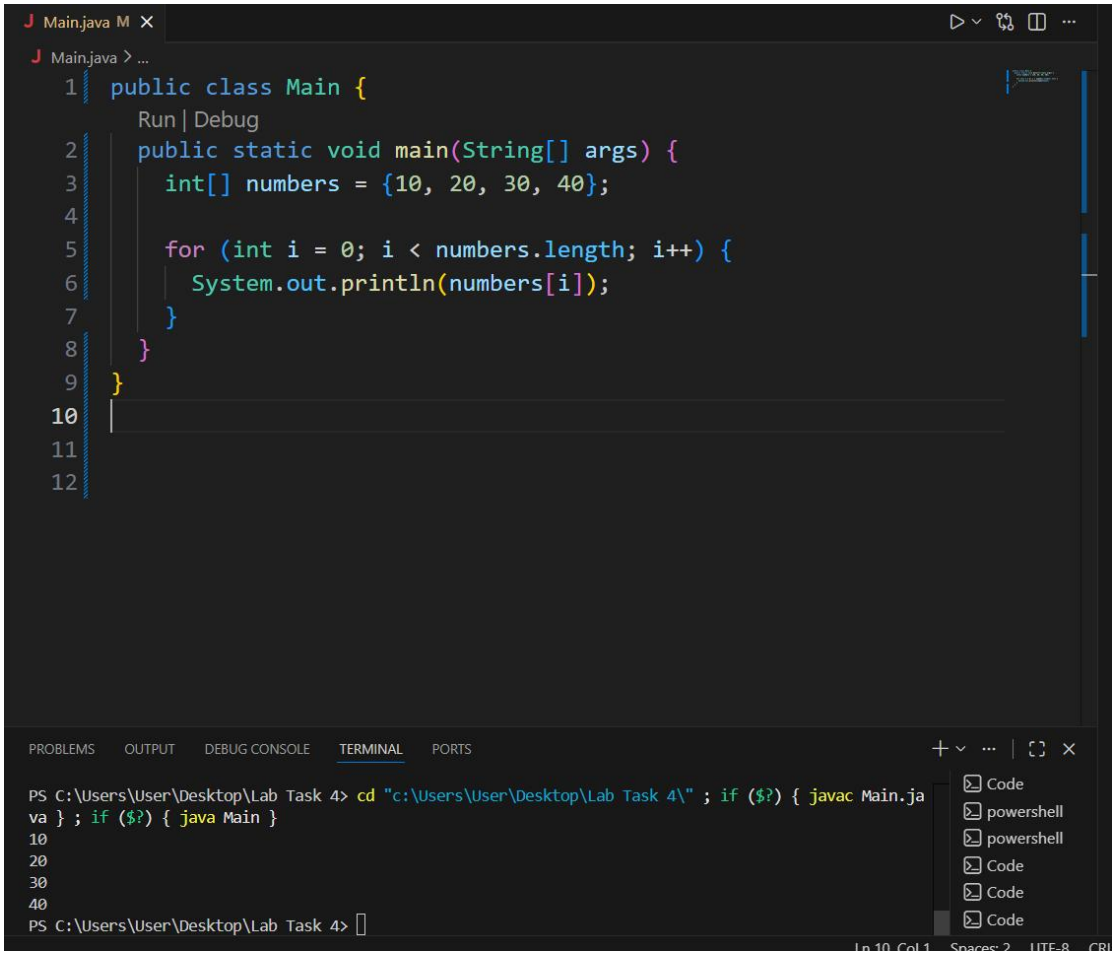
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array named `numbers` is created with the values `{10, 20, 30, 40}`.

The program then uses a **for loop** to go through each element of the array:

The loop starts with `i = 0` and continues as long as `i < numbers.length`.

Inside the loop, `numbers[i]` prints the element at index `i`.

This demonstrates how a `for` loop can be used to **access and print all elements of a number array**.



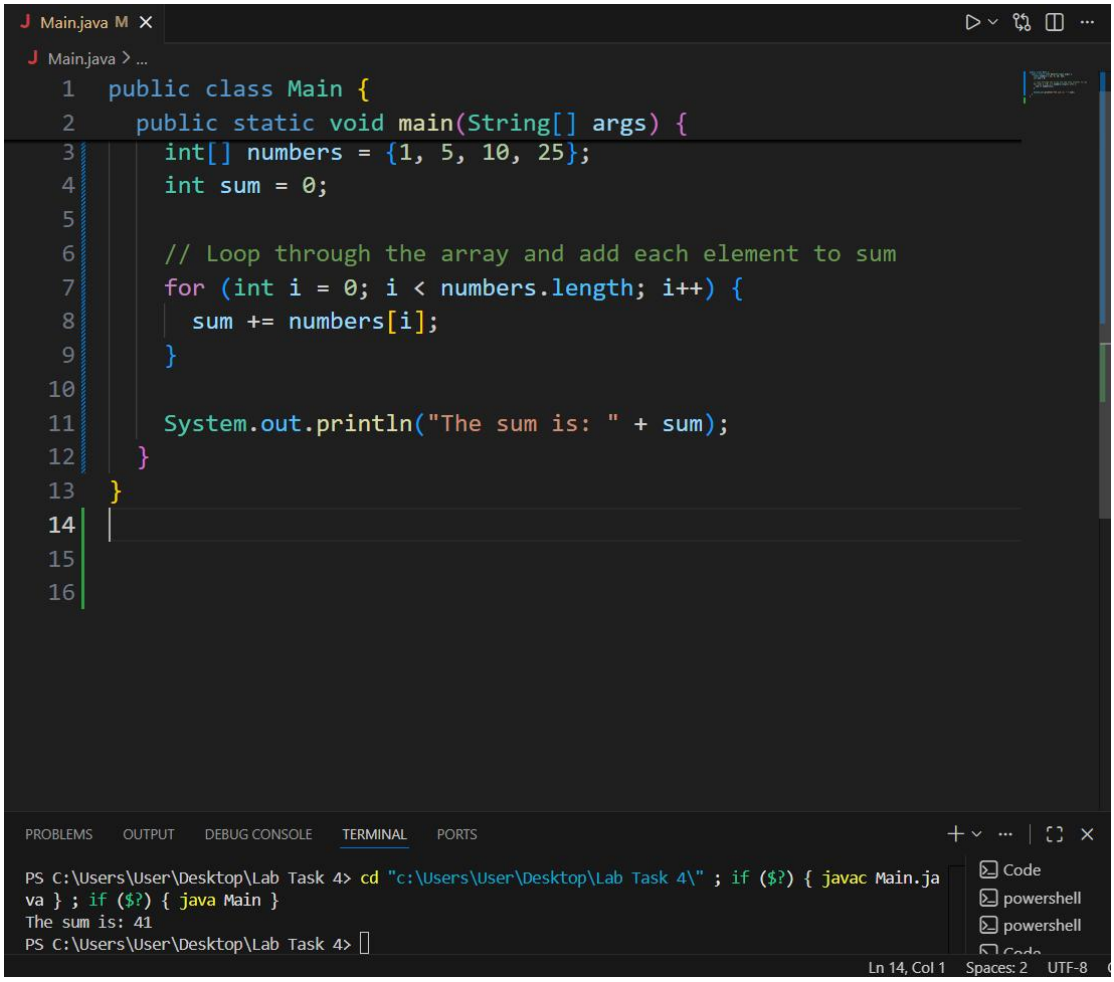
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array named `numbers` is created with the values `{1, 5, 10, 25}`.

The program then calculates the **sum of all elements** in the array using a `for` loop:

- A variable `sum` is initialized to `0`.
- The loop goes through each element of the array using `i` from `0` to `numbers.length - 1`.
- In each iteration, the current element `numbers[i]` is added to `sum` using `sum += numbers[i]`.

Finally, the program prints the total sum using `System.out.println()`.

This demonstrates how to loop through an array and calculate the sum of its elements.



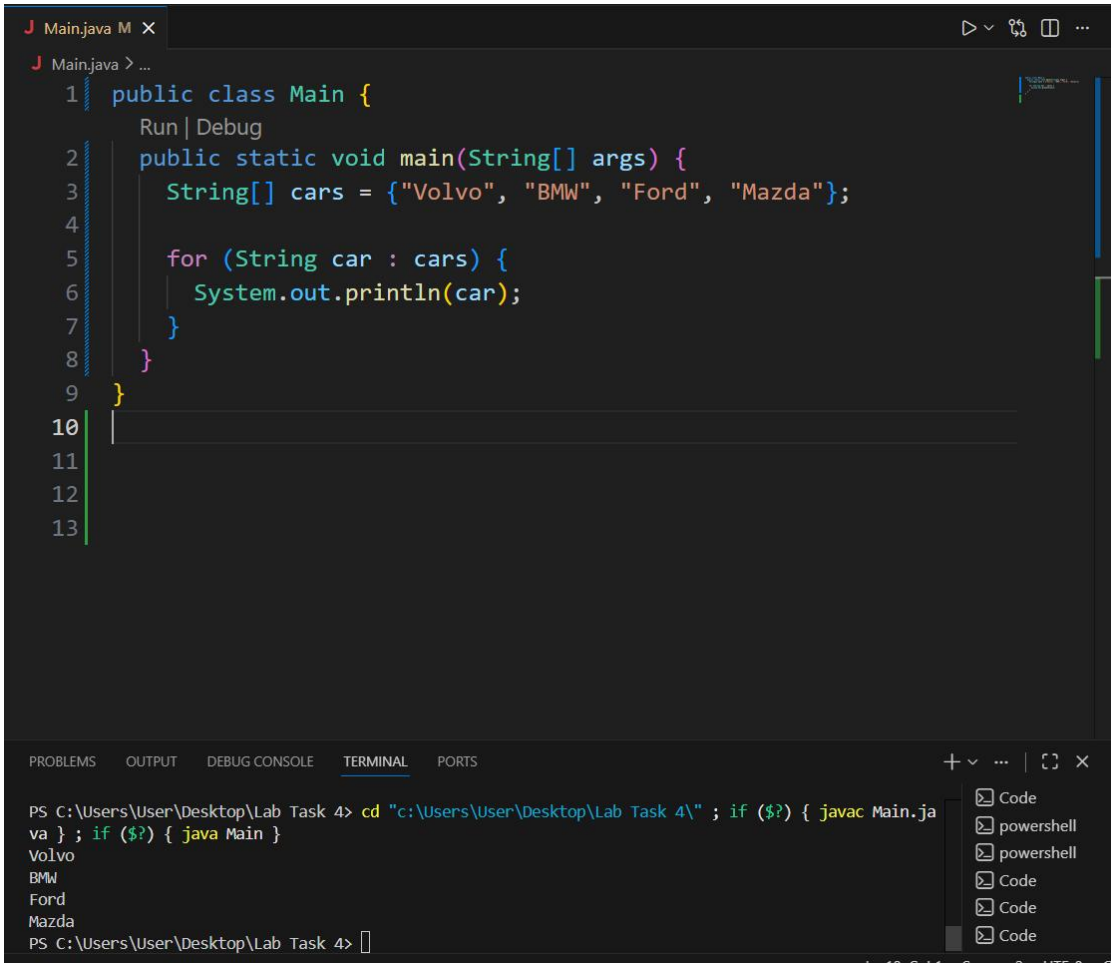
This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `cars` is created with the values `"Volvo"`, `"BMW"`, `"Ford"`, and `"Mazda"`.

The program then uses a **for-each loop** (`for (String car : cars)`) to go through each element of the array:

In each iteration, the current element is stored in the variable `car`.

`System.out.println(car)` prints that element.

This demonstrates how a **for-each loop** can be used to **easily access and print all elements of an array**.

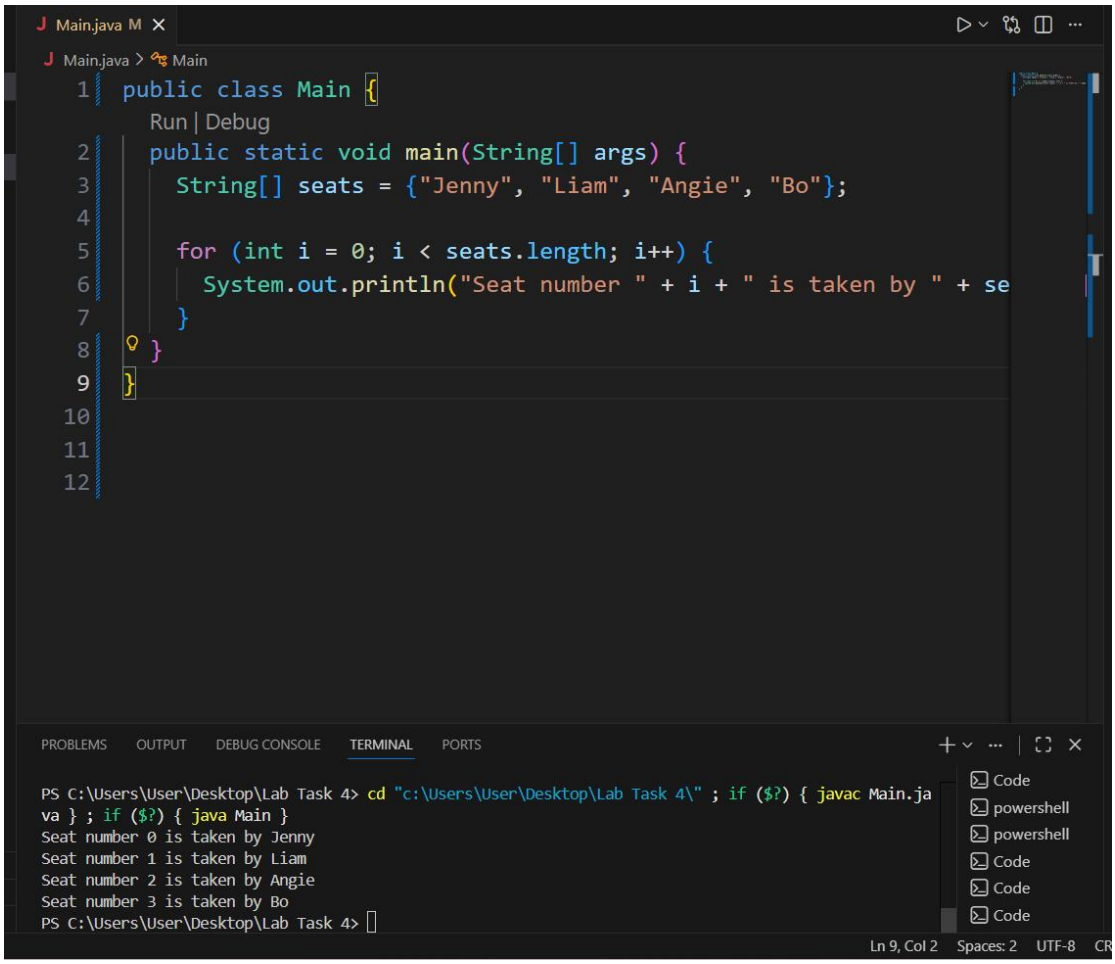


This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array named `seats` is created with the values `"Jenny"`, `"Liam"`, `"Angie"`, and `"Bo"`.

The program then uses a **for loop** to go through each element of the array:

> The loop starts with `i = 0` and continues as long as `i < seats.length`.
>
> In each iteration, the program prints the seat number (`i`) and the name of the person in that seat (`seats[i]`) using `System.out.println()`.

This shows how to use a loop to access array elements along with their index.



JAVA Arrays Real Life Examples

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array `ages` is created with values `{20, 22, 18, 35, 48, 26, 87, 70}`.

The program calculates the **average age** using the following steps:

A variable `sum` is initialized to `0` to store the total of all ages.

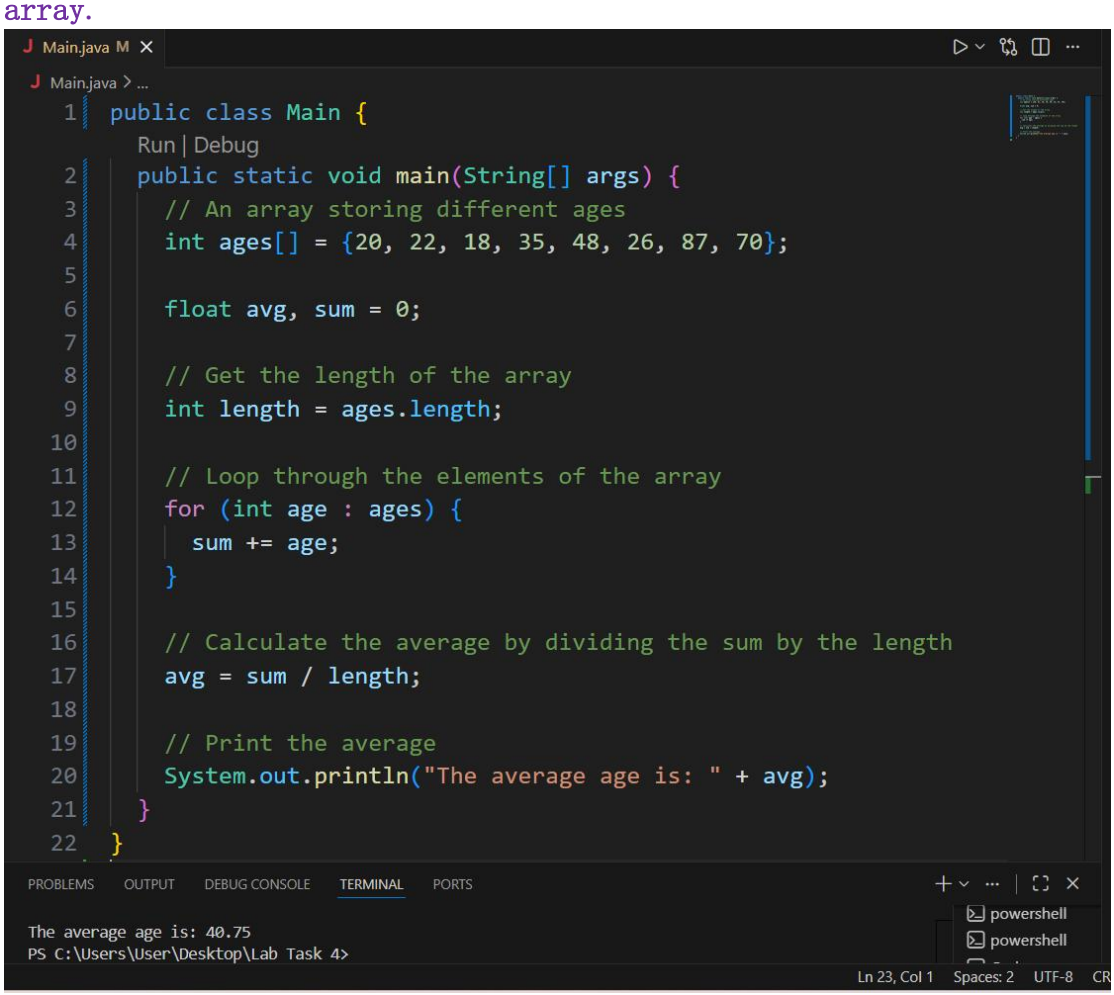
The program finds the length of the array using `ages.length`.

A **for-each loop** goes through each element (`age`) of the array, adding each value to `sum`.

The average is calculated by dividing `sum` by the number of elements (`length`).

Finally, the program prints the average using `System.out.println()`.

This demonstrates how to **calculate the average of all elements in an array.**



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array `ages` is created with the values `{20, 22, 18, 35, 48, 26, 87, 70}`.

The program finds the **lowest age** in the array using the following steps:

It gets the length of the array using `ages.length` (though it's not strictly needed here).

A variable `lowestAge` is initialized with the first element of the array (`ages[0]`).

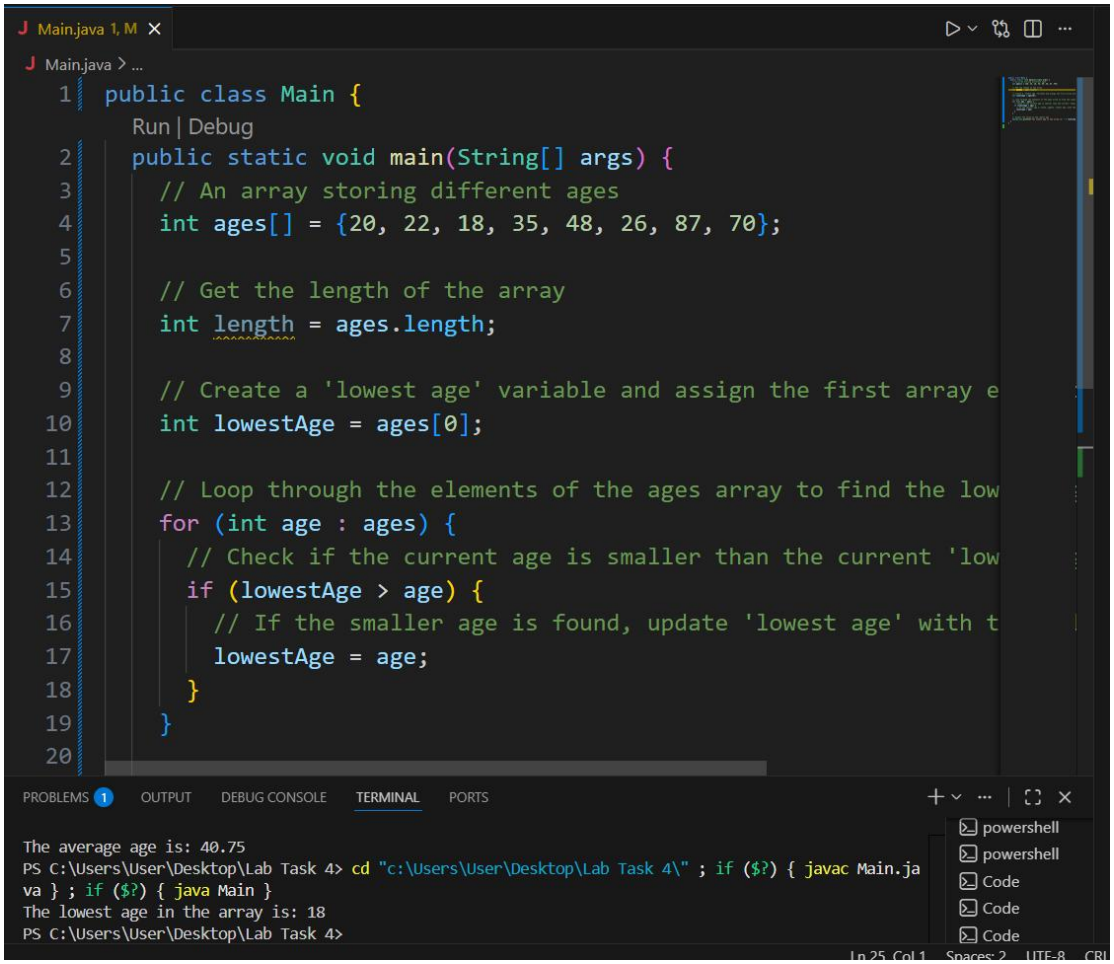
A **for-each loop** goes through each `age` in the array.

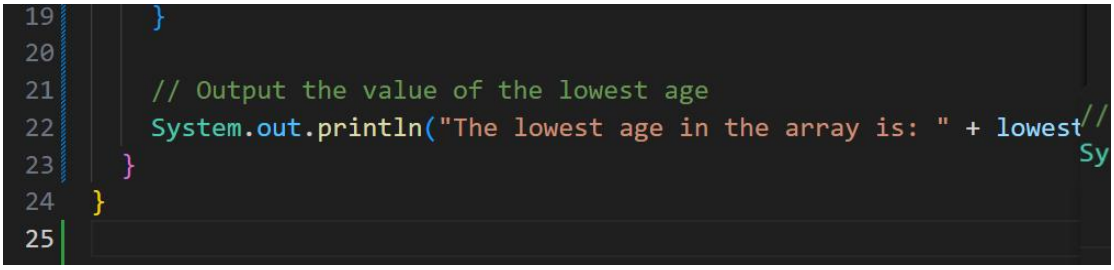
Inside the loop, the program checks if the current `age` is smaller than `lowestAge`.

If true, `lowestAge` is updated to this smaller value.

After the loop finishes, `lowestAge` contains the smallest value in the array, which is then printed.

This demonstrates how to **find the smallest element in an array** using a loop.


Main.java 1, M
Main.java > ...
1 public class Main {
Run | Debug
2 public static void main(String[] args) {
3 // An array storing different ages
4 int ages[] = {20, 22, 18, 35, 48, 26, 87, 70};
5
6 // Get the length of the array
7 int length = ages.length;
8
9 // Create a 'lowest age' variable and assign the first array e
10 int lowestAge = ages[0];
11
12 // Loop through the elements of the ages array to find the low
13 for (int age : ages) {
14 // Check if the current age is smaller than the current 'low
15 if (lowestAge > age) {
16 // If the smaller age is found, update 'lowest age' with t
17 lowestAge = age;
18 }
19 }
20
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
The average age is: 40.75
PS C:\Users\User\Desktop\Lab Task 4> cd "c:\Users\User\Desktop\Lab Task 4\" ; if ($?) { javac Main.ja
va } ; if ($?) { java Main }
The lowest age in the array is: 18
PS C:\Users\User\Desktop\Lab Task 4>
powershell
powershell
Code
Code
Code

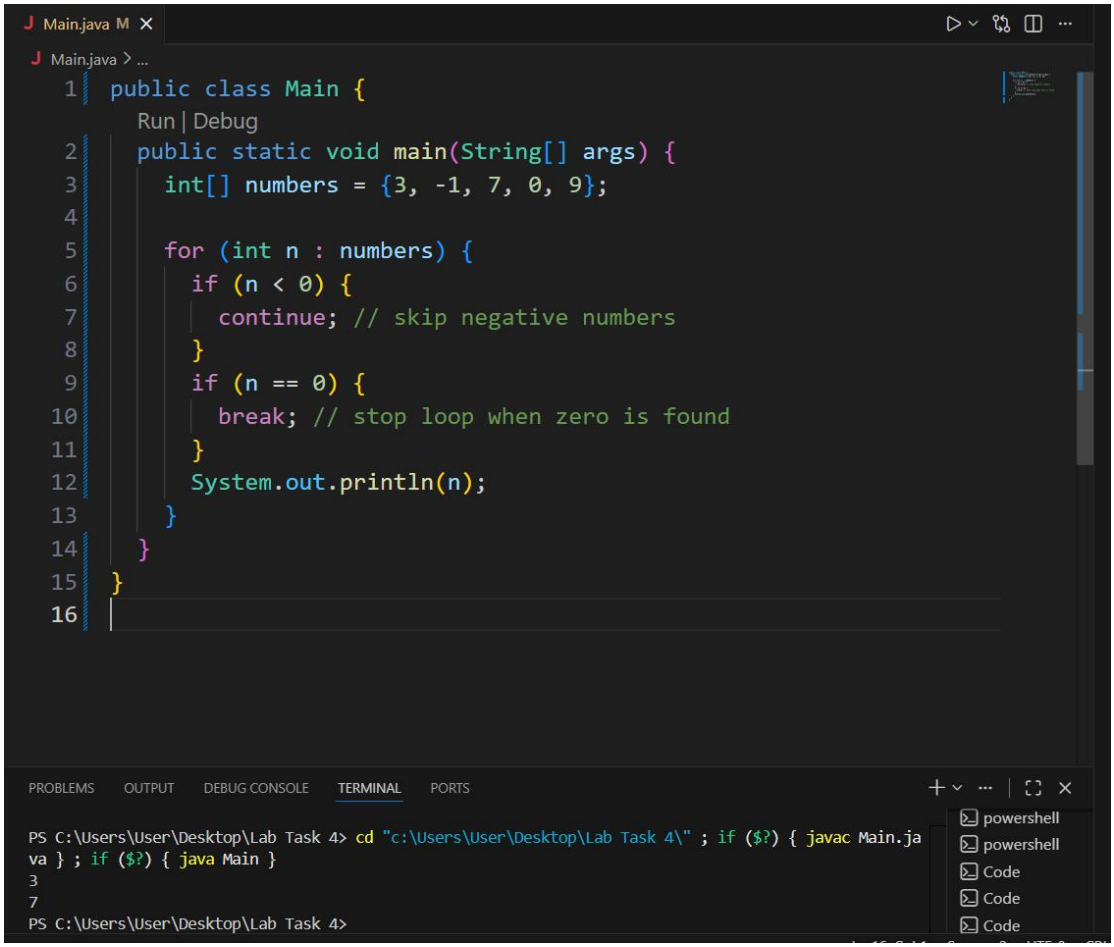

}
// Output the value of the lowest age
System.out.println("The lowest age in the array is: " + lowest
}
}

This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array `numbers` is created with the values `{3, -1, 7, 0, 9}`.

The program uses a **for-each loop** to go through each number in the array:

> If the current number `n` is **negative** (`n < 0`), the `continue` statement **skips that number** and moves to the next iteration.
>
> If the current number `n` is **zero** (`n == 0`), the `break` statement **stops the loop completely**.
>
> Otherwise, the number is printed using `System.out.println(n)`.

This demonstrates how continue can skip certain values and break can stop the loop early.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, an integer array `numbers` is created with the values `{45, 12, 98, 33, 27}`.

The program finds both the **maximum** and **minimum** values in the array using these steps:

Two variables `max` and `min` are initialized with the first element of the array (`numbers[0]`).

A **for-each loop** goes through each element `n` in the array.

Inside the loop:

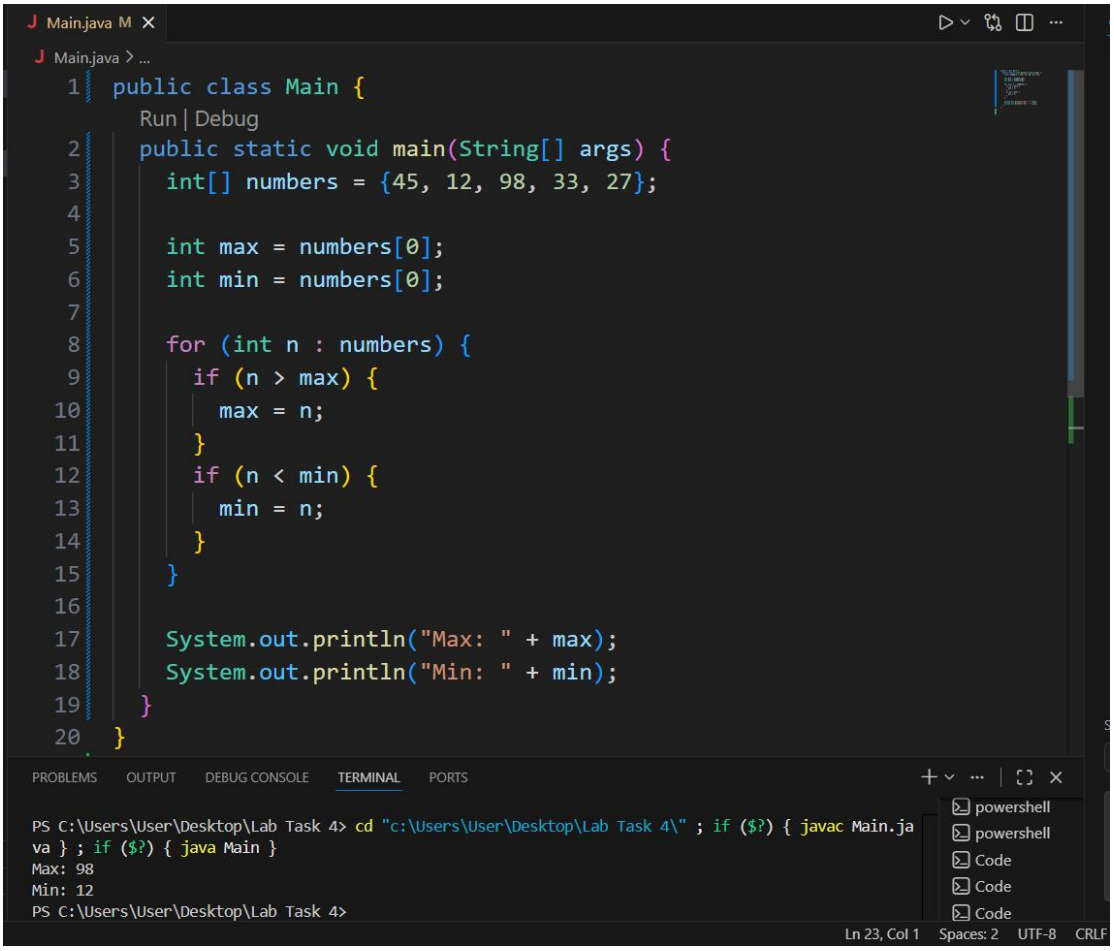
If `n > max`, `max` is updated to `n`.

If `n < min`, `min` is updated to `n`.

After the loop finishes, `max` contains the largest value, and `min` contains the smallest value.

The program prints both `max` and `min` using `System.out.println()`.

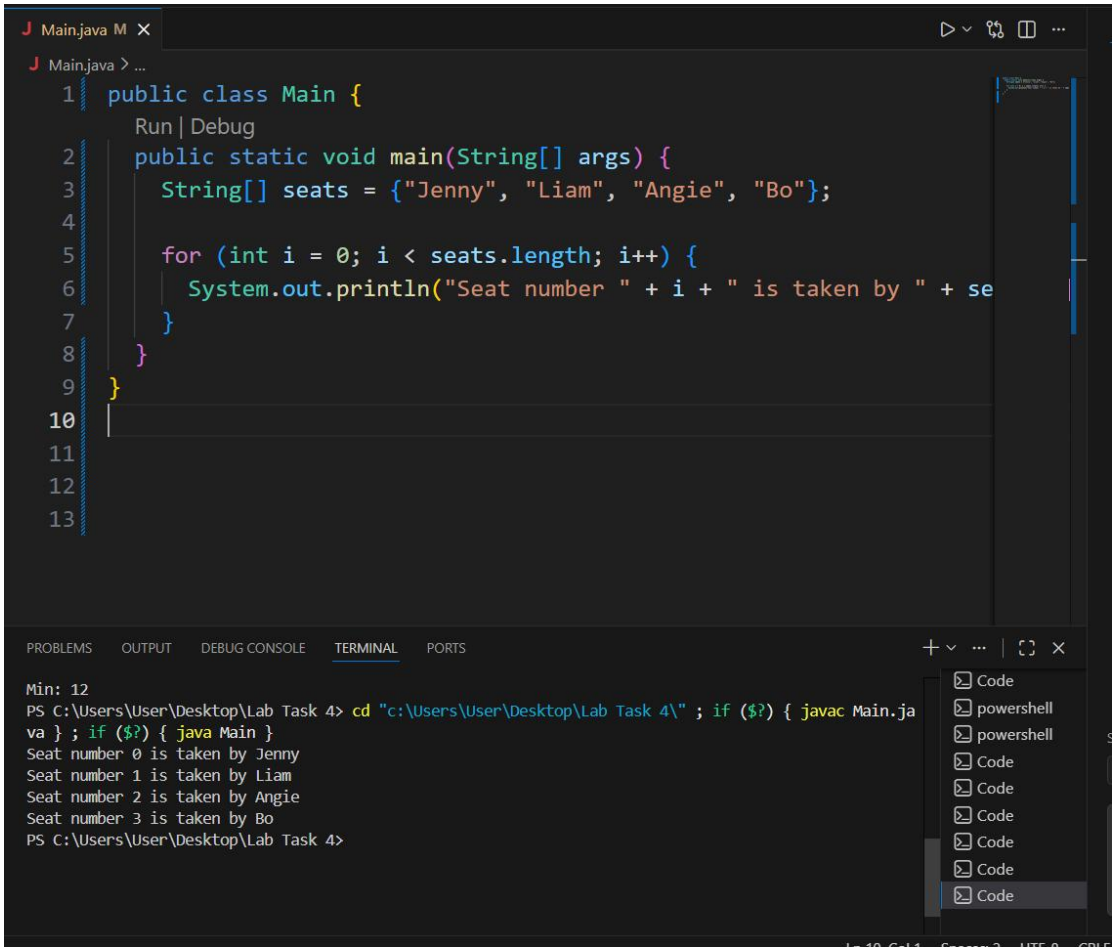
This demonstrates how to **find the largest and smallest elements in an array** using a loop.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a string array `seats` is created with the values `"Jenny"`, `"Liam"`, `"Angie"`, and `"Bo"`.

The program uses a **for loop** to go through each element of the array:

> The loop starts with `i = 0` and continues as long as `i < seats.length`.
>
> Inside the loop, the program prints the seat number (`i`) and the person sitting in that seat (`seats[i]`) using `System.out.println()`.


Main.java M
Main.java > ...
1 public class Main {
Run | Debug
2 public static void main(String[] args) {
3 String[] seats = {"Jenny", "Liam", "Angie", "Bo"};
4
5 for (int i = 0; i < seats.length; i++) {
6 System.out.println("Seat number " + i + " is taken by " + se
7 }
8 }
9 }
10
11
12
13
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Min: 12
PS C:\Users\User\Desktop\Lab Task 4> cd "c:\Users\User\Desktop\Lab Task 4\" ; if ($?) { javac Main.ja
va } ; if ($?) { java Main }
Seat number 0 is taken by Jenny
Seat number 1 is taken by Liam
Seat number 2 is taken by Angie
Seat number 3 is taken by Bo
PS C:\Users\User\Desktop\Lab Task 4>
Code
powershell
powershell
Code
Code
Code
Code
Code
Code

Java Multidimensional Array

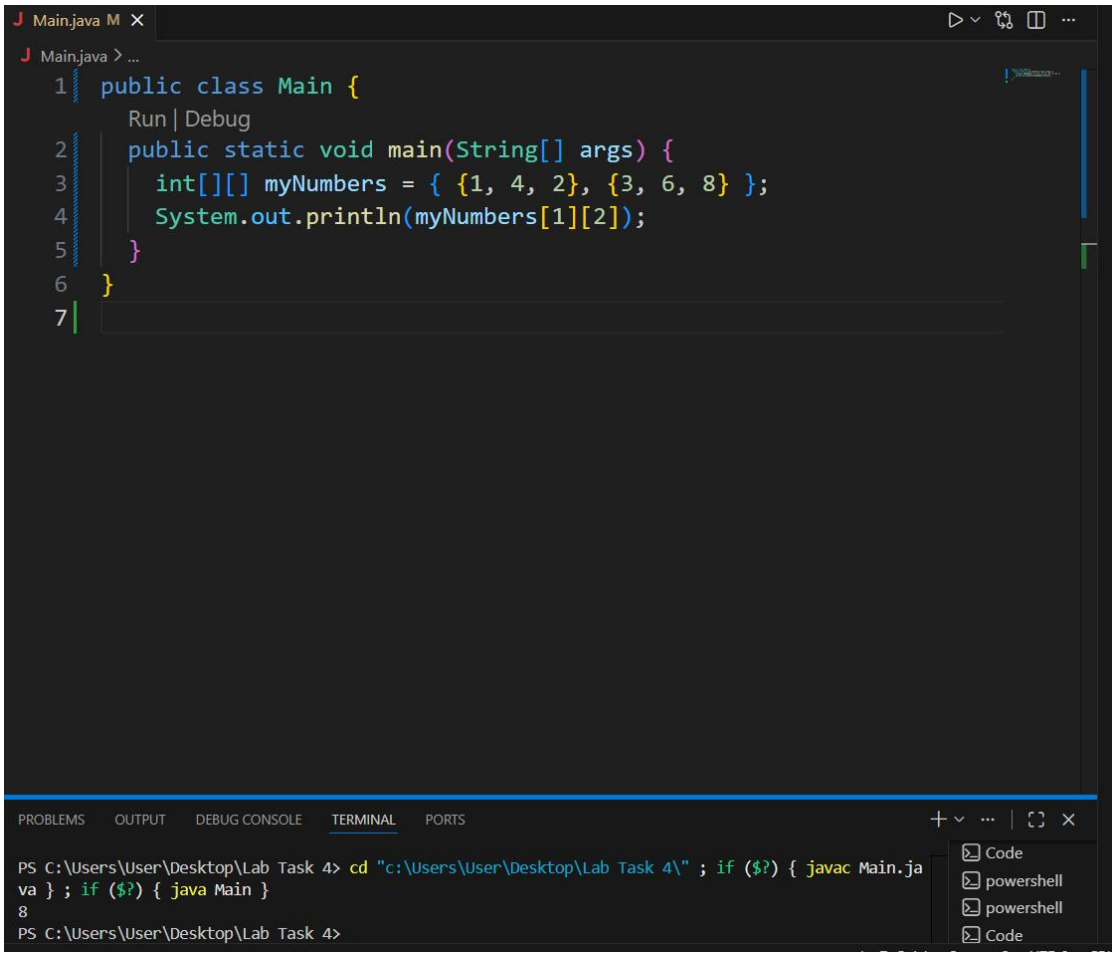
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a **2D array** myNumbers is created with two rows:

The program accesses a specific element of the 2D array using **row and column indices**:

`myNumbers[1][2]` means **row 1, column 2** (remember indexing starts at 0).

In this array, row 1 is `{3, 6, 8}`, so column 2 is `8`.

Finally, it prints the value `8` using `System.out.println()`.



This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a 2D array myNumbers is created with two rows:

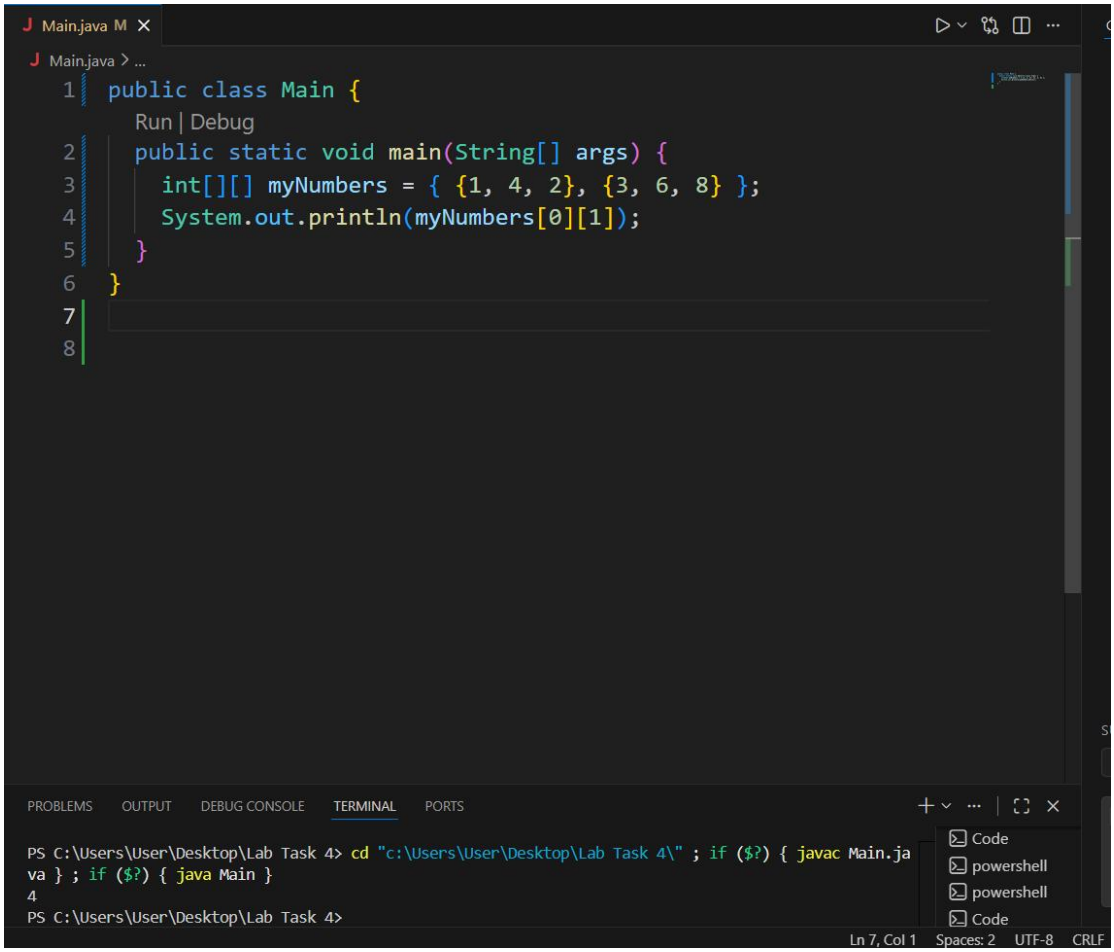
The program accesses a specific element of the 2D array using **row and column indices**:

`myNumbers[0][1]` means **row 0, column 1** (remember indexing starts at 0).

In this array, row 0 is `{1, 4, 2}`, so column 1 is `4`.

Finally, it prints the value `4` using `System.out.println()`.

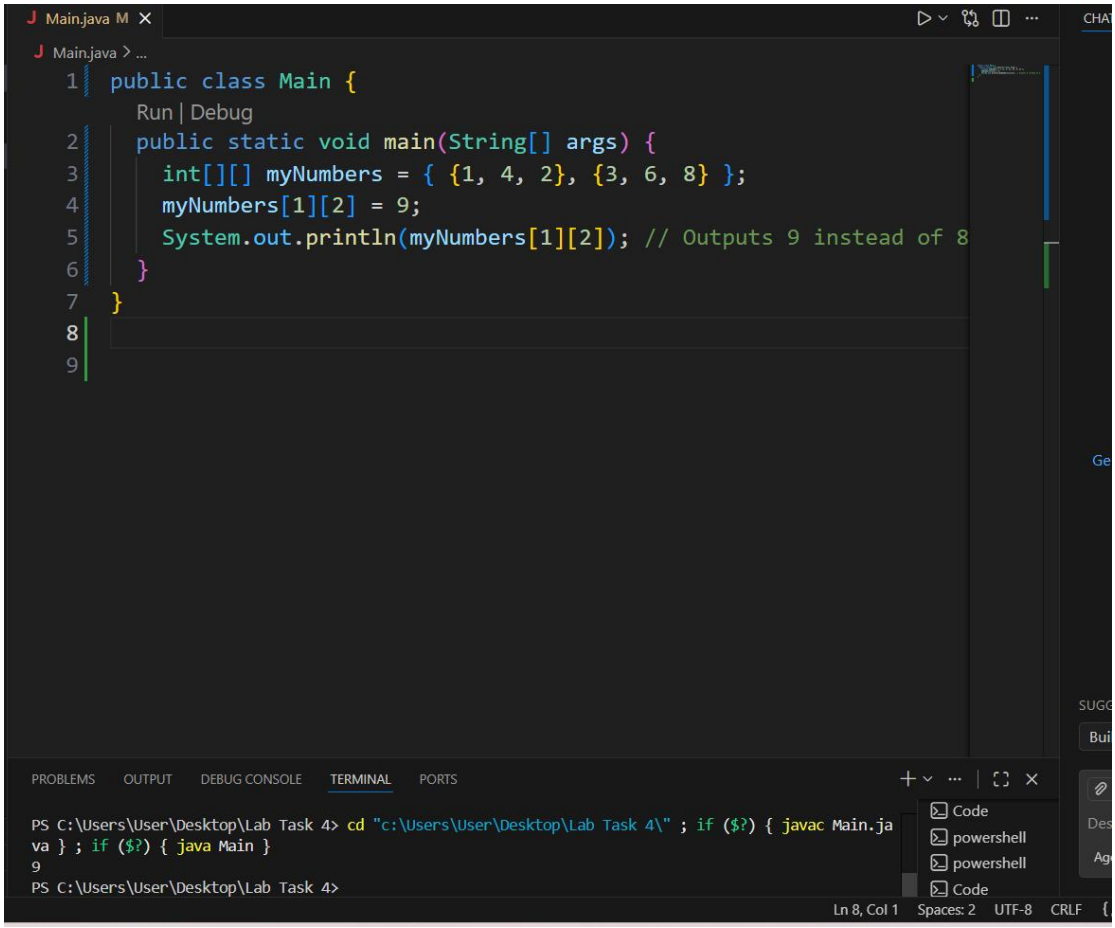
This demonstrates how to access elements in a two-dimensional array using row and column indices.



This program defines a class named `Main`, and the `main` method is where the program starts executing. Inside the `main` method, a **2D array** `myNumbers` is created with two rows:

The program then **changes a specific element** of the array using its **row and column indices**:

> `myNumbers[1][2] = 9;` updates the value at **row 1, column 2** (which was `8`) to `9`.
>
> `System.out.println(myNumbers[1][2]);` prints this updated value.
>
> This demonstrates how to **modify an element in a two-dimensional array** using its row and column indices.



This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a **2D array** myNumbers is created with **two rows**:

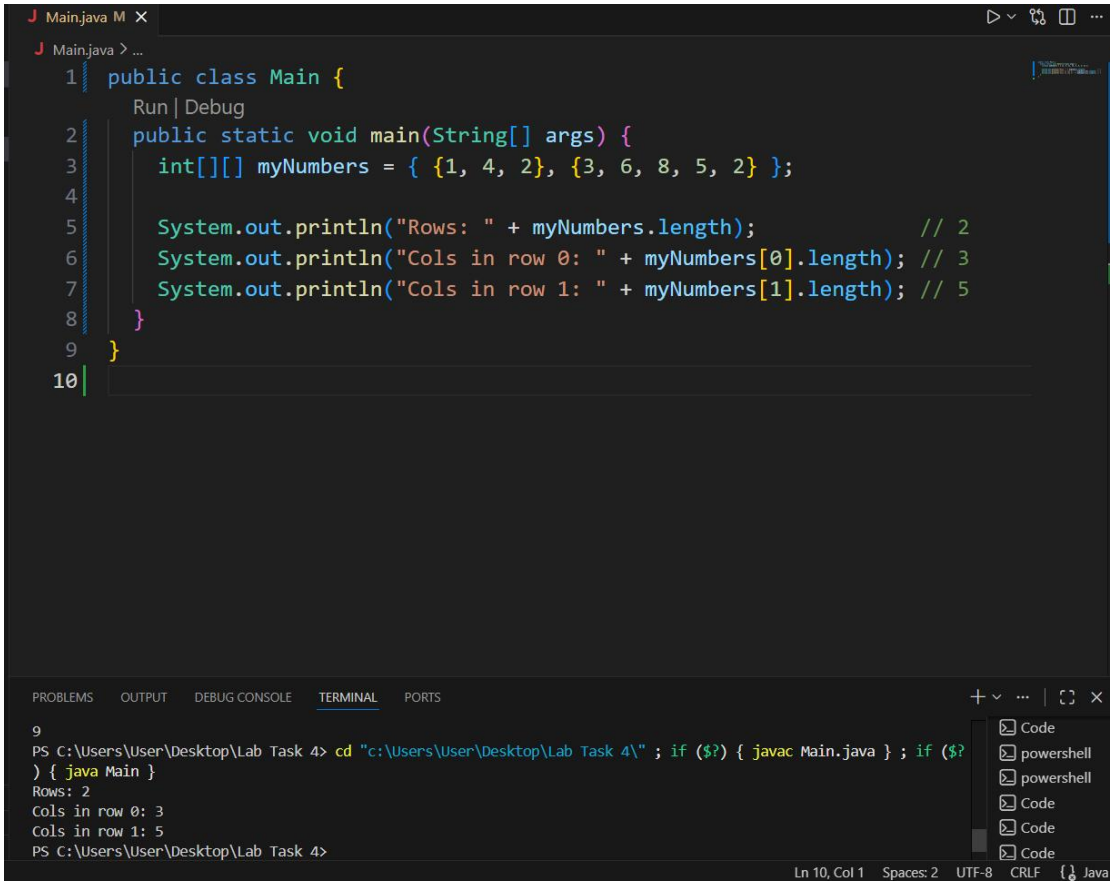
The program then prints information about the **dimensions of the 2D array**:

`myNumbers.length` gives the **number of rows** in the array, which is `2`.

`myNumbers[0].length` gives the **number of columns in row 0**, which is `3`.

`myNumbers[1].length` gives the **number of columns in row 1**, which is `5`.

This demonstrates that **2D arrays in Java can have rows with different lengths** (also called jagged arrays) and how to check the number of rows and columns.

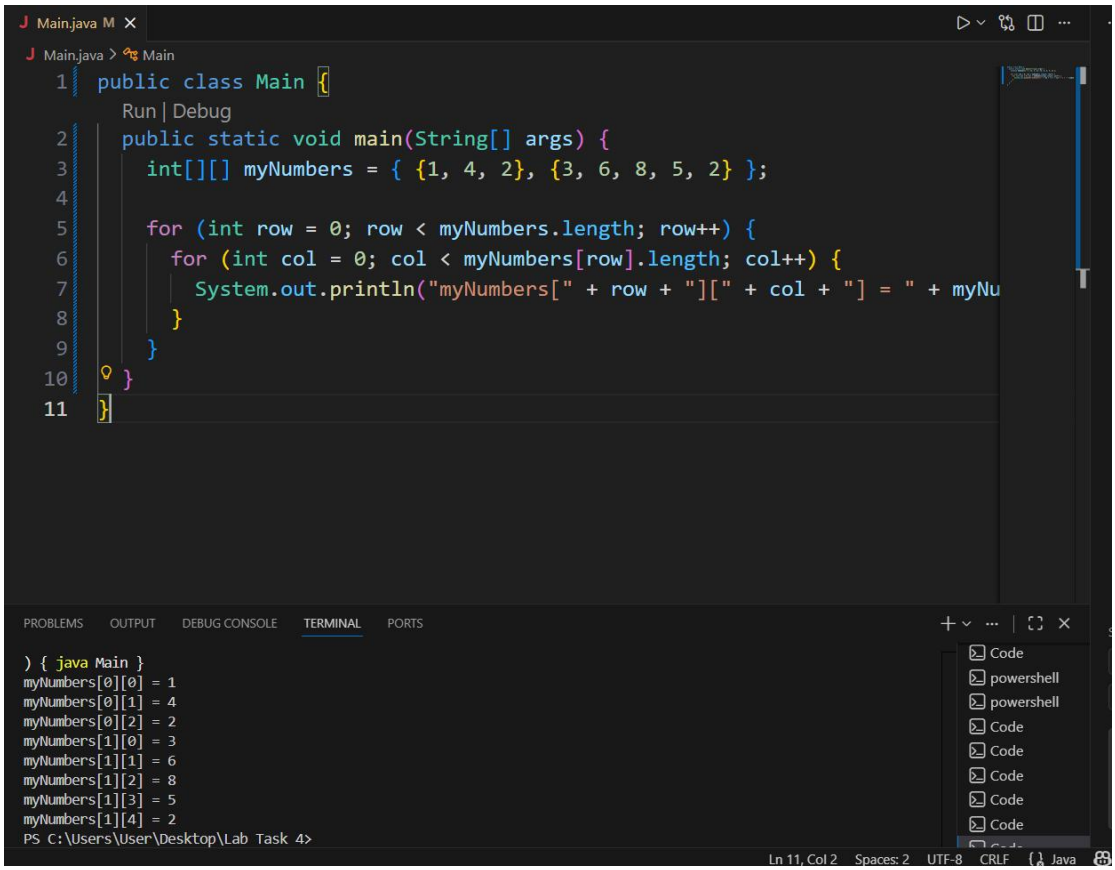


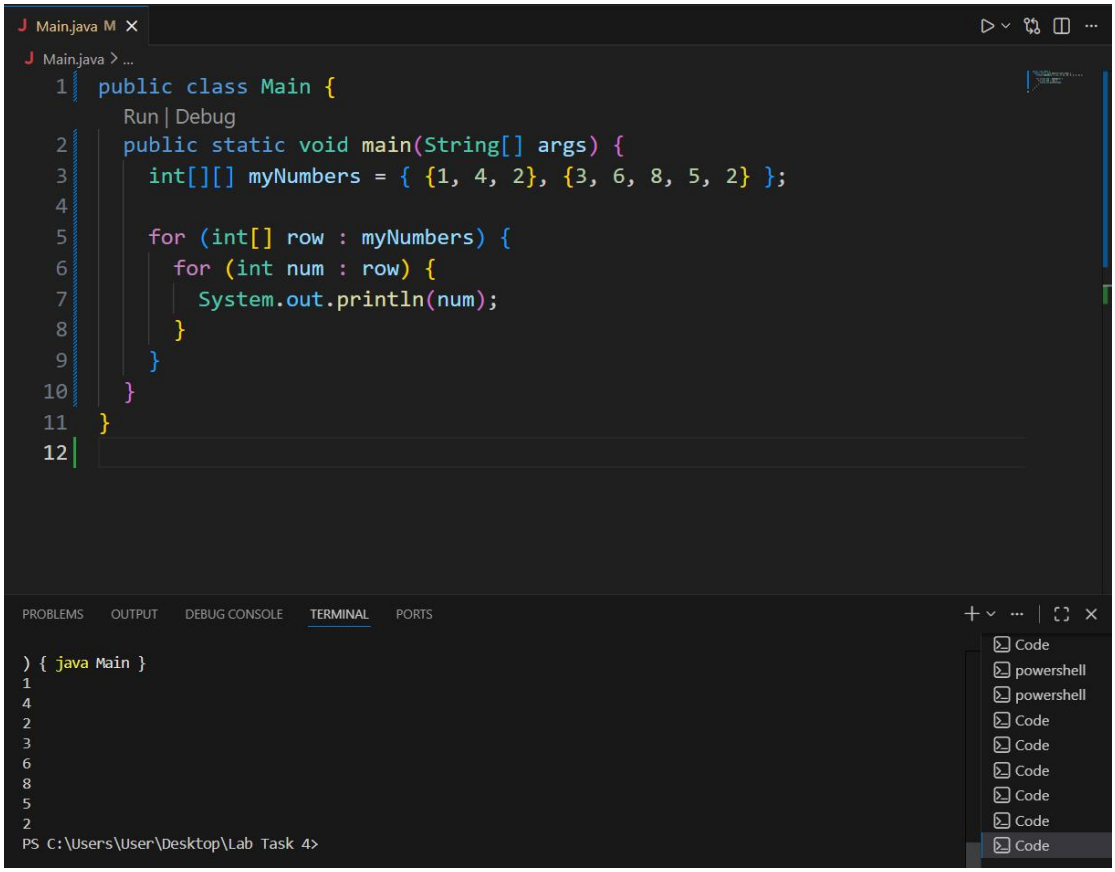
This program defines a class named Main, and the main method is where the program starts executing. Inside the main method, a 2D array myNumbers is created with two rows:

The program uses **nested for-loops** to print all the elements of the 2D array:

- The **outer loop** iterates through the rows (`row = 0` to `myNumbers.length - 1`).
- The **inner loop** iterates through the columns of each row (`col = 0` to `myNumbers[row].length - 1`).
- Inside the inner loop, each element is printed along with its **row and column indices**.

This demonstrates how to traverse and print all elements of a two-dimensional (possibly jagged) array using nested loops.




Main.java M
Main.java > ...
1 public class Main {
Run | Debug
2 public static void main(String[] args) {
3 int[][] myNumbers = { {1, 4, 2}, {3, 6, 8, 5, 2} };
4
5 for (int[] row : myNumbers) {
6 for (int num : row) {
7 System.out.println(num);
8 }
9 }
10 }
11 }
12
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
) { java Main }
1
4
2
3
6
8
5
2
PS C:\Users\User\Desktop\Lab Task 4>
Code
powershell
powershell
Code
Code
Code
Code
Code
Code