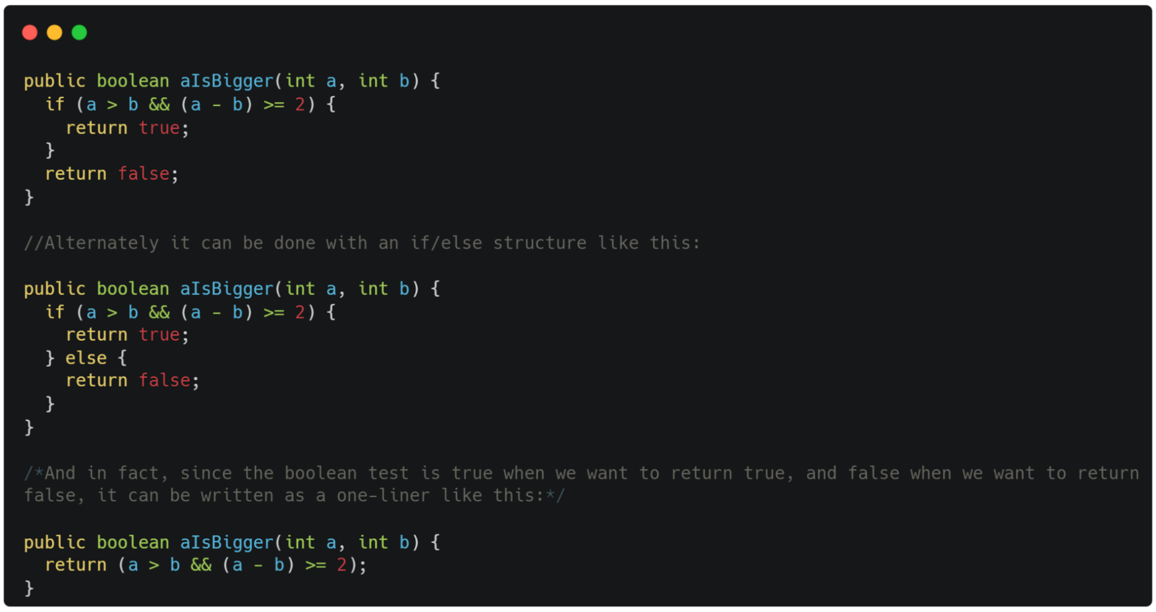
CodingBat code practice

If-Boolean Logic

In this example, we have a method called aIsBigger() that checks if the value of an integer parameter a is greater than the value of another integer b by 2 or more.

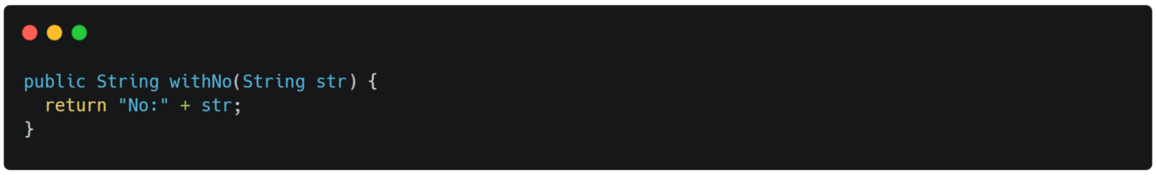


```java
public boolean aIsBigger(int a, int b) {
  if (a > b && (a - b) >= 2) {
    return true;
  }
  return false;
}

//Alternately it can be done with an if/else structure like this:

public boolean aIsBigger(int a, int b) {
  if (a > b && (a - b) >= 2) {
    return true;
  } else {
    return false;
  }
}

/*And in fact, since the boolean test is true when we want to return true, and false when we want to return
false, it can be written as a one-liner like this:*/

public boolean aIsBigger(int a, int b) {
  return (a > b && (a - b) >= 2);
}
```

Strings

Text can be enclosed in double quotes to create string literals. Let's explore some common string techniques:

1. **Concatenating Strings :**
   Strings can be enclosed in double quotes to create string literals. You can use the + operator to combine strings



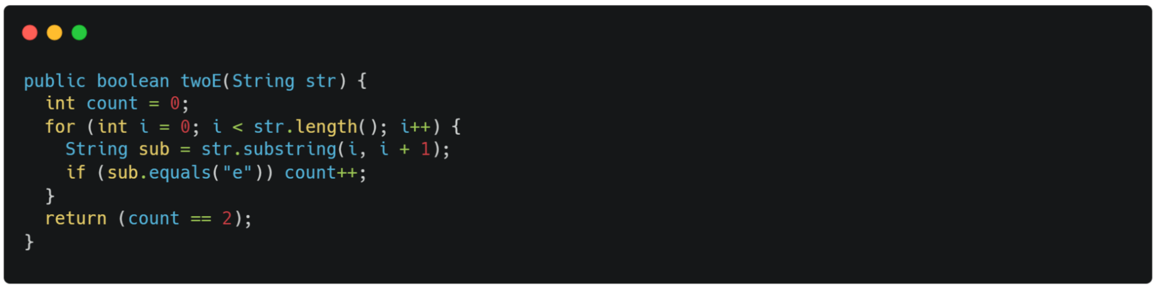
```java
public String withNo(String str) {
  return "No:" + str;
}
```

2. **Substrings and String Length:**
   The substring(i, j) method extracts a substring starting from index i up to (but not including) index j. The length() method returns the length of a string.
3. **Comparing Strings:**
   To compare the content of strings, use .equals() instead of “ = = ”.
4. **Looping through String Characters:**
   A standard approach to iterate through each character in a string is using a loop.
5. **toUpperCase() / toLowerCase():**
   Converts the string to uppercase or lowercase.
6. **startsWith(prefix) / endsWith(suffix):**
   Checks if the string starts with the specified prefix or ends with the specified suffix.

Counting Occurrences of a Character:

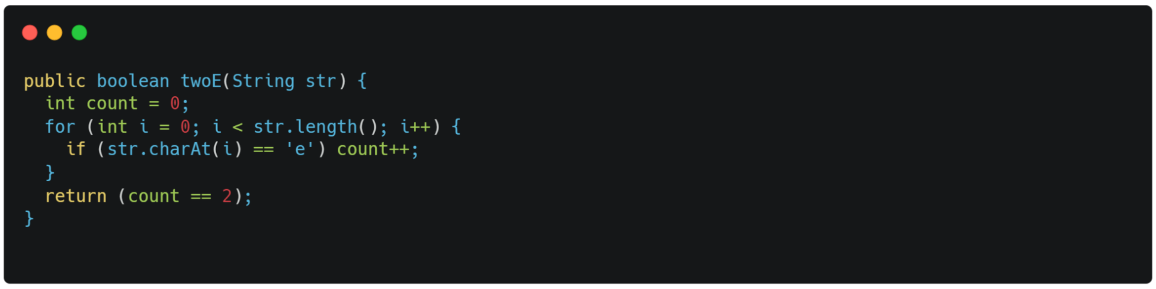
The twoE() method counts occurrences of the character 'e' in a string.

Using substring():



```java
public boolean twoE(String str) {
  int count = 0;
  for (int i = 0; i < str.length(); i++) {
    String sub = str.substring(i, i + 1);
    if (sub.equals("e")) count++;
  }
  return (count == 2);
}
```

Using charAt():



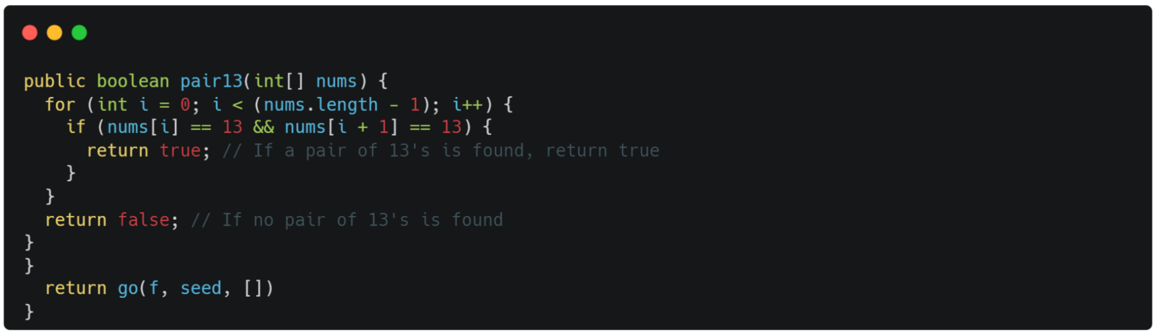
```java
public boolean twoE(String str) {
  int count = 0;
  for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) == 'e') count++;
  }
  return (count == 2);
}
```

Arrays

To declare an array, you specify the data type of the elements followed by the array variable name and square brackets [ ].

pair13() Method:

This method, pair13(), checks whether an integer array contains a pair of consecutive 13 values.

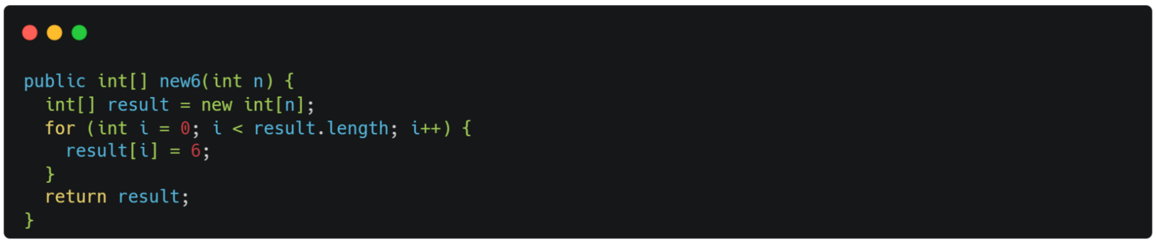


```java
public boolean pair13(int[] nums) {
  for (int i = 0; i < (nums.length - 1); i++) {
    if (nums[i] == 13 && nums[i + 1] == 13) {
      return true; // If a pair of 13's is found, return true
    }
  }
  return false; // If no pair of 13's is found
}
}
  return go(f, seed, [])
}
```

The method iterates through the array and checks each pair of consecutive values. If it encounters a pair of 13 values, it immediately returns true. If it goes through the entire array without finding a pair of 13 values, it returns false.

Note: The loop stops one element short of the full length of the array to safely access nums[i+1] within the loop.

new6() Method:
The new6() method creates and returns a new integer array of size N, where all elements are filled with the value 6.

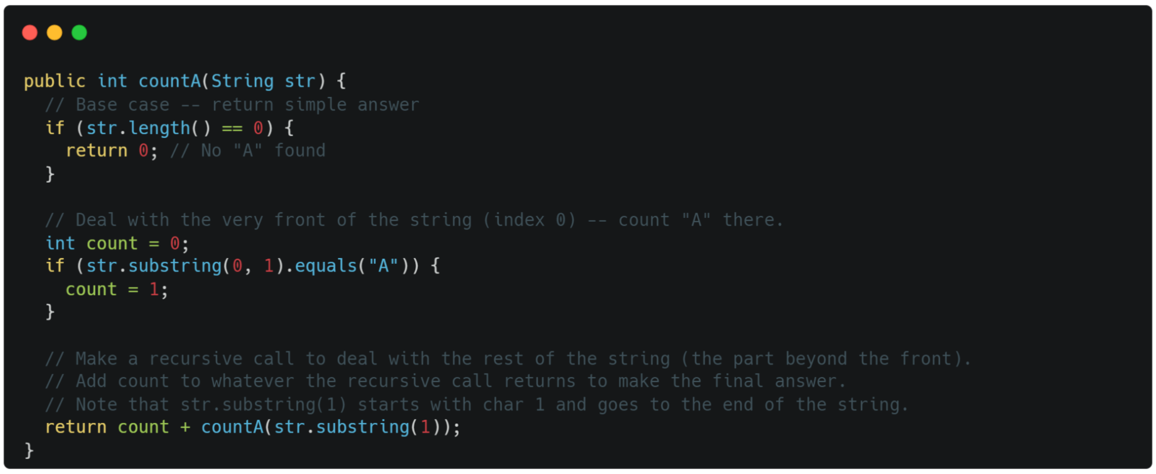


```java
public int[] new6(int n) {
  int[] result = new int[n];
  for (int i = 0; i < result.length; i++) {
    result[i] = 6;
  }
  return result;
}
```

This method initialises a new integer array named result with the specified size n. Then, it iterates through the array and sets each element to the value 6. Finally, it returns the result array with all elements initialised to 6.

Recursion:

Recursion is a programming technique where a function calls itself to solve a problem. It involves breaking down a complex problem into simpler sub-problems that can be solved in a similar way.
Recursive Method Example: countA()
Here's an example of a recursive method that counts the number of occurrences of the character "A" in a given string.

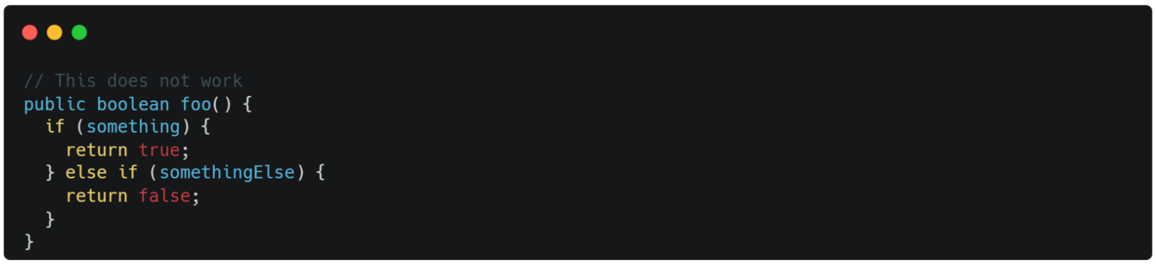


```java
public int countA(String str) {
  // Base case -- return simple answer
  if (str.length() == 0) {
    return 0; // No "A" found
  }

  // Deal with the very front of the string (index 0) -- count "A" there.
  int count = 0;
  if (str.substring(0, 1).equals("A")) {
    count = 1;
  }

  // Make a recursive call to deal with the rest of the string (the part beyond the front).
  // Add count to whatever the recursive call returns to make the final answer.
  // Note that str.substring(1) starts with char 1 and goes to the end of the string.
  return count + countA(str.substring(1));
}
```

Base Case: If the input string is empty, return 0 as there are no occurrences of "A".
Recursive Case: Count the "A" at the beginning of the string and make a recursive call with the rest of the string. Add the count of "A" in the substring to the count returned from the recursive call. The method keeps breaking down the problem into smaller subproblems until it reaches the base case.

Must Return Type X:

In Java, when a method is declared to return a specific type (e.g., boolean, int, String), every possible execution path within the method must lead to a return statement that provides a result of that type. This ensures that the method always returns a value of the expected type.
Compile Error Example:
Here's an example where the compiler raises an error due to a potential exit from the method without a return statement:

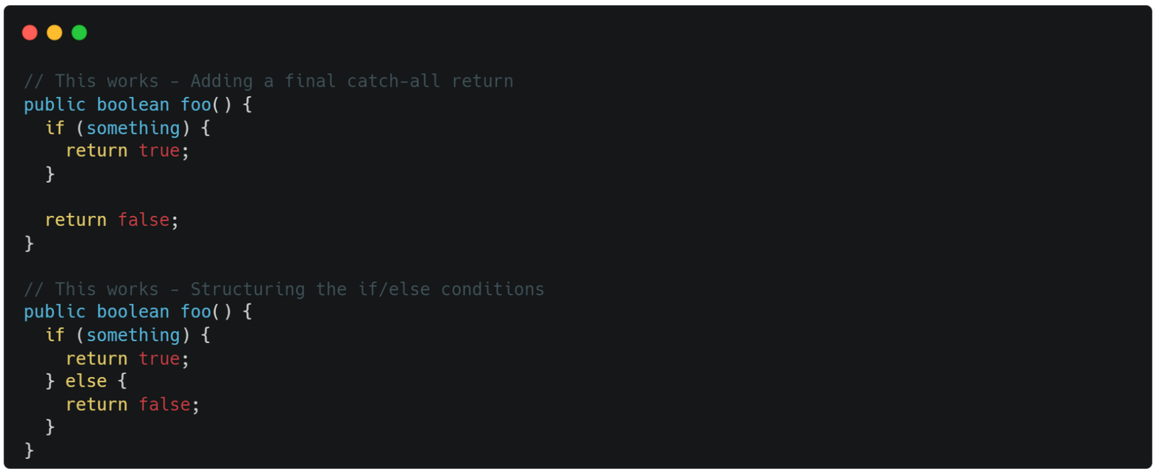


```java
// This does not work
public boolean foo() {
  if (something) {
    return true;
  } else if (somethingElse) {
    return false;
  }
}
```

In this case, if both ‘ if ’ conditions are false, the method will exit without a return statement, leading to a compile error.

Solutions:

To fix this error, you need to ensure that every possible path in the method leads to a return statement that provides a result of the expected type. You can do this by adding a final catch-all return statement, or by structuring the if/else conditions properly:



```java
// This works - Adding a final catch-all return
public boolean foo() {
  if (something) {
    return true;
  }

  return false;
}

// This works - Structuring the if/else conditions
public boolean foo() {
  if (something) {
    return true;
  } else {
    return false;
  }
}
```

Both of these solutions ensure that the method will always return a boolean value, regardless of the execution path.