**Explanation:**

1. **Main Method:**

java

Copy code

public static void main(String[] args) {

String name1 = "Dawod";

String name2 = "woadD";

System.out.println(isAnagram(name1, name2));

}

* + Defines two strings, name1 and name2.
  + Calls the isAnagram method with these strings and prints the result.

1. **Anagram Check Method:**

java

Copy code

public static boolean isAnagram(String str1, String str2) {

// Check if the lengths are different

if (str1.length() != str2.length()) {

return false;

}

// Create maps to store the frequency of characters

Map<Character, Integer> charCountMap1 = buildCharCountMap(str1);

Map<Character, Integer> charCountMap2 = buildCharCountMap(str2);

// Compare the two maps

return charCountMap1.equals(charCountMap2);

}

* + First checks if the lengths of the two strings are different. If they are, they can't be anagrams.
  + Builds frequency maps for both strings using the buildCharCountMap method.
  + Compares the two frequency maps. If they are equal, the strings are anagrams.

1. **Build Frequency Map Method:**

java

Copy code

private static Map<Character, Integer> buildCharCountMap(String str) {

Map<Character, Integer> charCountMap = new HashMap<>();

for (char c : str.toCharArray()) {

charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);

}

return charCountMap;

}

* + Converts the string into a character array and iterates over each character.
  + Updates the count of each character in the HashMap. Uses getOrDefault to simplify the count update.

**Advantages of This Approach:**

1. **Efficiency:** The time complexity of this approach is O(n), where n is the length of the strings. This is efficient because each string is scanned once to build the frequency maps.
2. **Clarity:** The use of maps makes it clear how character frequencies are counted and compared.
3. **Handling Different Cases:** This approach handles cases where characters are in different cases by counting each character based on its ASCII value.

This method is often preferred for its clarity and efficiency, especially when dealing with large strings.