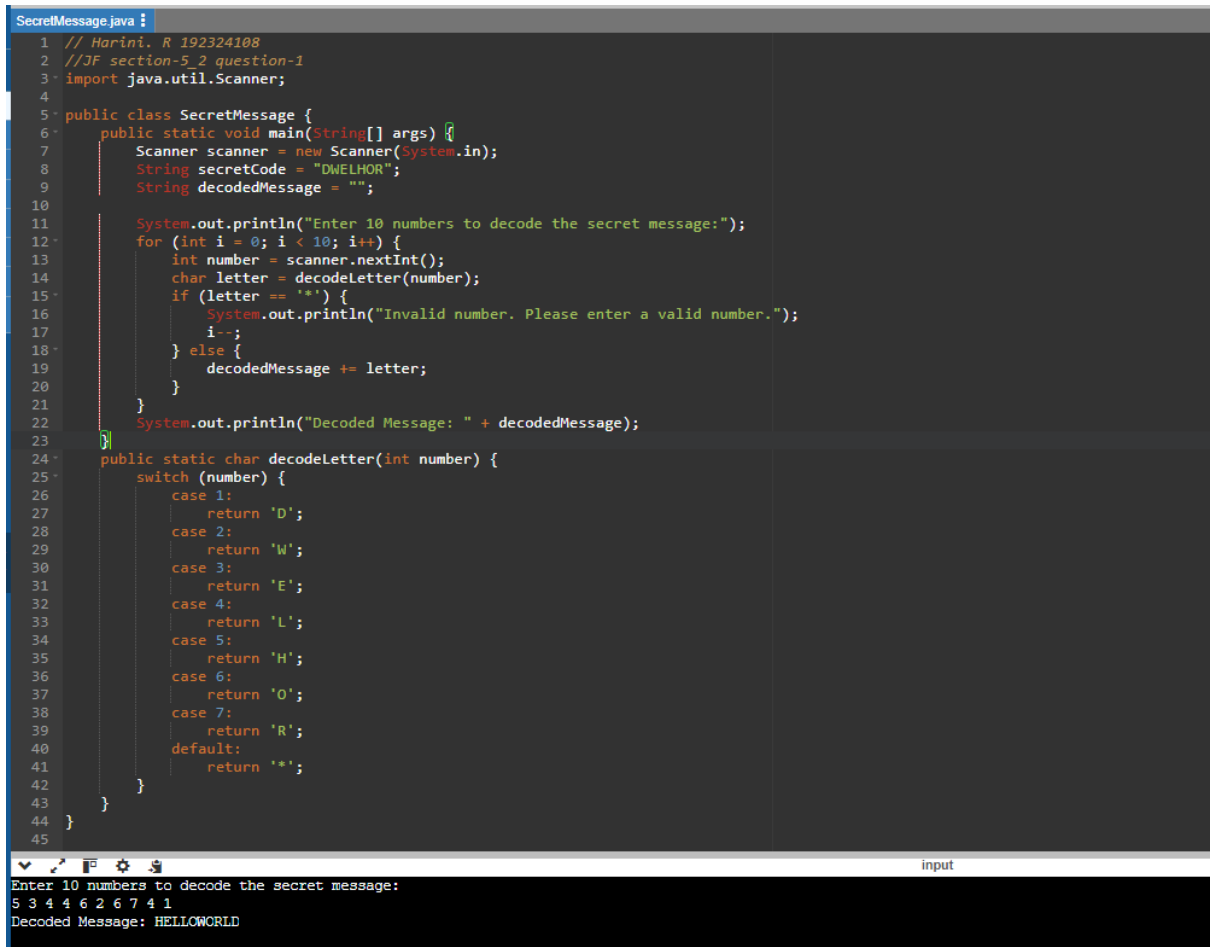


Java Fundamentals 5_2

NAME: HARINI.R

REGISTER NO: 192324108

1. Write a program that prompts the user for 10 numbers, one at a time, and prints out the decoded message. If the user enters a number that is not one of those already deciphered, prompt him/her for a new number. Test your code with the following input:
5 3 4 4 6 2 6 7 4 1



```
SecretMessage.java :
1 // Harini. R 192324108
2 //JF section-5_2 question-1
3 import java.util.Scanner;
4
5 public class SecretMessage {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         String secretCode = "DHELHOR";
9         String decodedMessage = "";
10
11         System.out.println("Enter 10 numbers to decode the secret message:");
12         for (int i = 0; i < 10; i++) {
13             int number = scanner.nextInt();
14             char letter = decodeLetter(number);
15             if (letter == '*') {
16                 System.out.println("Invalid number. Please enter a valid number.");
17                 i--;
18             } else {
19                 decodedMessage += letter;
20             }
21         }
22         System.out.println("Decoded Message: " + decodedMessage);
23     }
24     public static char decodeLetter(int number) {
25         switch (number) {
26             case 1:
27                 return 'D';
28             case 2:
29                 return 'H';
30             case 3:
31                 return 'E';
32             case 4:
33                 return 'L';
34             case 5:
35                 return 'H';
36             case 6:
37                 return 'O';
38             case 7:
39                 return 'R';
40             default:
41                 return '*';
42         }
43     }
44 }
45
```

input

Enter 10 numbers to decode the secret message:
5 3 4 4 6 2 6 7 4 1
Decoded Message: HELLOWORLD

2. Suppose you are implementing a search routine that searches through a String, character by character, until it finds a space character. As soon as you find the first space character, you decide that you do not want to continue searching the string. If you are using a WHILE loop and your loop will continue to execute until you have gone through the entire string, should you use the keyword break or continue when you find the first space character? Why? Why would you not use the other keyword?

ANSWER:

When implementing a search routine in Java that searches through a `String` character by character using a `while` loop, you should use the keyword `break` when you find the first space character.

Explanation:

- ``break``: The ``break`` keyword immediately terminates the loop when a condition is met. In this case, as soon as you find the first space character, you want to stop searching through the string because your goal has been achieved. By using ``break``, the loop exits immediately, and no further iterations are performed.
- ``continue``: The ``continue`` keyword, on the other hand, skips the current iteration and proceeds with the next iteration of the loop. If you use ``continue`` when you find the first space character, the loop will skip the remaining statements in the current iteration and move on to check the next character in the string. This means the loop would continue to execute even after finding the space, which is not what you want.

Summary:

- Use ``break``: It stops the loop as soon as the first space character is found, which is the desired behavior.
- Don't use ``continue``: It would skip to the next iteration, causing the loop to continue even after the space character is found, which is unnecessary and inefficient.

3. Imagine you are writing a program that prints out the day of the week (Sunday, Monday, Tuesday, etc.) for each day of the year. Before the program executes, can you tell how many times the loop will execute? Assume the year is not a Leap year. Given your answer, which type of loop would you need to implement? Explain your reasoning.

ANSWER:

Yes, you can determine how many times the loop will execute before the program runs. Since the year is not a leap year, it has 365 days. Therefore, the loop will execute 365 times, once for each day of the year.

Reasoning:

- Number of Executions: Since the program needs to print the day of the week for each day of the year, and there are 365 days in a non-leap year, the loop will execute exactly 365 times.
- Type of Loop: Given that the number of iterations is known beforehand, a ``for`` loop would be the most appropriate choice.

Why Use a ``for`` Loop:

- Fixed Iterations: A ``for`` loop is ideal when the number of iterations is predetermined. In this case, you know in advance that the loop should run 365 times, so a ``for`` loop provides a clear, concise, and readable way to handle this scenario.

- Syntax Clarity: The ``for`` loop allows you to define the initialization, condition, and increment in a single line, making it clear how many times the loop will run.

```
for (int day = 1; day <= 365; day++) {  
    // Logic to determine and print the day of the week  
}
```

Here, ``day`` starts at 1 and increments by 1 until it reaches 365, ensuring the loop runs exactly 365 times.

Why Not Use Other Loops:

- ``while`` Loop: While you could use a ``while`` loop, it's typically better suited for situations where the number of iterations is not known in advance. A ``while`` loop would require additional logic to manage the loop counter, making the code less concise.

- ``do-while`` Loop: A ``do-while`` loop guarantees that the loop body is executed at least once, which is unnecessary here since the number of iterations is fixed. Also, like the ``while`` loop, it might require extra logic to manage the loop counter.

Summary:

A ``for`` loop is the best choice because you know the loop needs to execute exactly 365 times, and the ``for`` loop provides a clear and concise structure for fixed iteration counts.

4. An anagram is a word or a phrase made by transposing the letters of another word or phrase; for example, "parliament" is an anagram of "partial men," and "software" is an anagram of "swear oft." Write a program that figures out whether one string is an anagram of another string. The program should ignore white space and punctuation.

```

1 // Harini. R 192324108
2 //JF section-5_2 question-4
3 import java.util.Arrays;
4
5 public class AnagramChecker {
6     public static void main(String[] args) {
7         String str1 = "parliament";
8         String str2 = "partial men";
9
10        boolean result = areAnagrams(str1, str2);
11        if (result) {
12            System.out.println("The strings are anagrams.");
13        } else {
14            System.out.println("The strings are not anagrams.");
15        }
16    }
17    public static boolean areAnagrams(String str1, String str2) {
18        String normalizedStr1 = normalizeString(str1);
19        String normalizedStr2 = normalizeString(str2);
20        char[] charArray1 = normalizedStr1.toCharArray();
21        char[] charArray2 = normalizedStr2.toCharArray();
22        Arrays.sort(charArray1);
23        Arrays.sort(charArray2);
24        return Arrays.equals(charArray1, charArray2);
25    }
26    private static String normalizeString(String str) {
27        return str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
28    }
29 }
30

```

The strings are anagrams.

input