SECTION 5.2

1. Consider you are asked to decode a secret message. The coded message is in numbers and each number stands for a specific letter. You discover enough of the secret code to decode the current message.

So far, you know:

• 1 represents “D”

• 2 represents “W”

• 3 represents “E”

• 4 represents “L”

• 5 represents “H”

• 6 represents “O”

• 7 represents “R”

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

**ANSWER:**

import java.util.Scanner;

public class SecretMessageDecoder {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Define the decoding map

char[] decodingMap = new char[8]; // index 0 is unused, indices 1-7 correspond to letters

decodingMap[1] = 'D';

decodingMap[2] = 'W';

decodingMap[3] = 'E';

decodingMap[4] = 'L';

decodingMap[5] = 'H';

decodingMap[6] = 'O';

decodingMap[7] = 'R';

StringBuilder decodedMessage = new StringBuilder();

System.out.println("Enter 10 numbers one by one to decode the message (valid numbers are 1 to 7):");

// Prompt the user to enter 10 numbers

for (int i = 0; i < 10; i++) {

int num = 0;

boolean validInput = false;

// Continue to prompt until a valid number is entered

while (!validInput) {

System.out.print("Enter number " + (i + 1) + ": ");

num = scanner.nextInt();

if (num >= 1 && num <= 7) {

validInput = true;

} else {

System.out.println("Invalid number. Please enter a number between 1 and 7.");

}

}

// Append the corresponding letter to the decoded message

decodedMessage.append(decodingMap[num]);

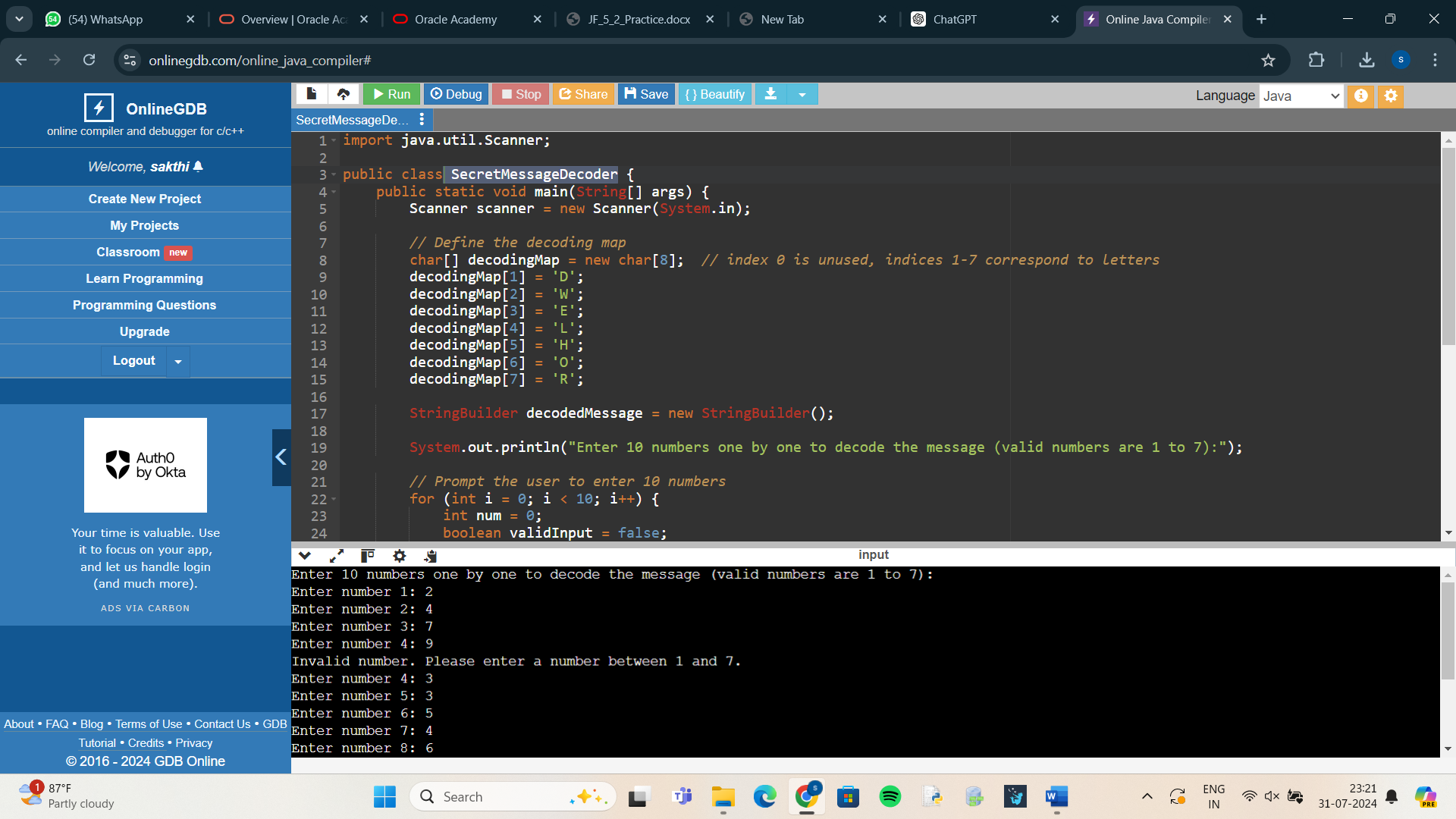
}

// Print the decoded message

System.out.println("The decoded message is: " + decodedMessage.toString());

}

}



1. 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:**

public class SearchString {

public static void main(String[] args) {

String str = "This is a test string";

int index = 0;

boolean foundSpace = false;

while (index < str.length()) {

char currentChar = str.charAt(index);

if (currentChar == ' ') {

foundSpace = true;

break; // Exit the loop immediately when a space is found

}

index++;

}

if (foundSpace) {

System.out.println("Space found at index " + index);

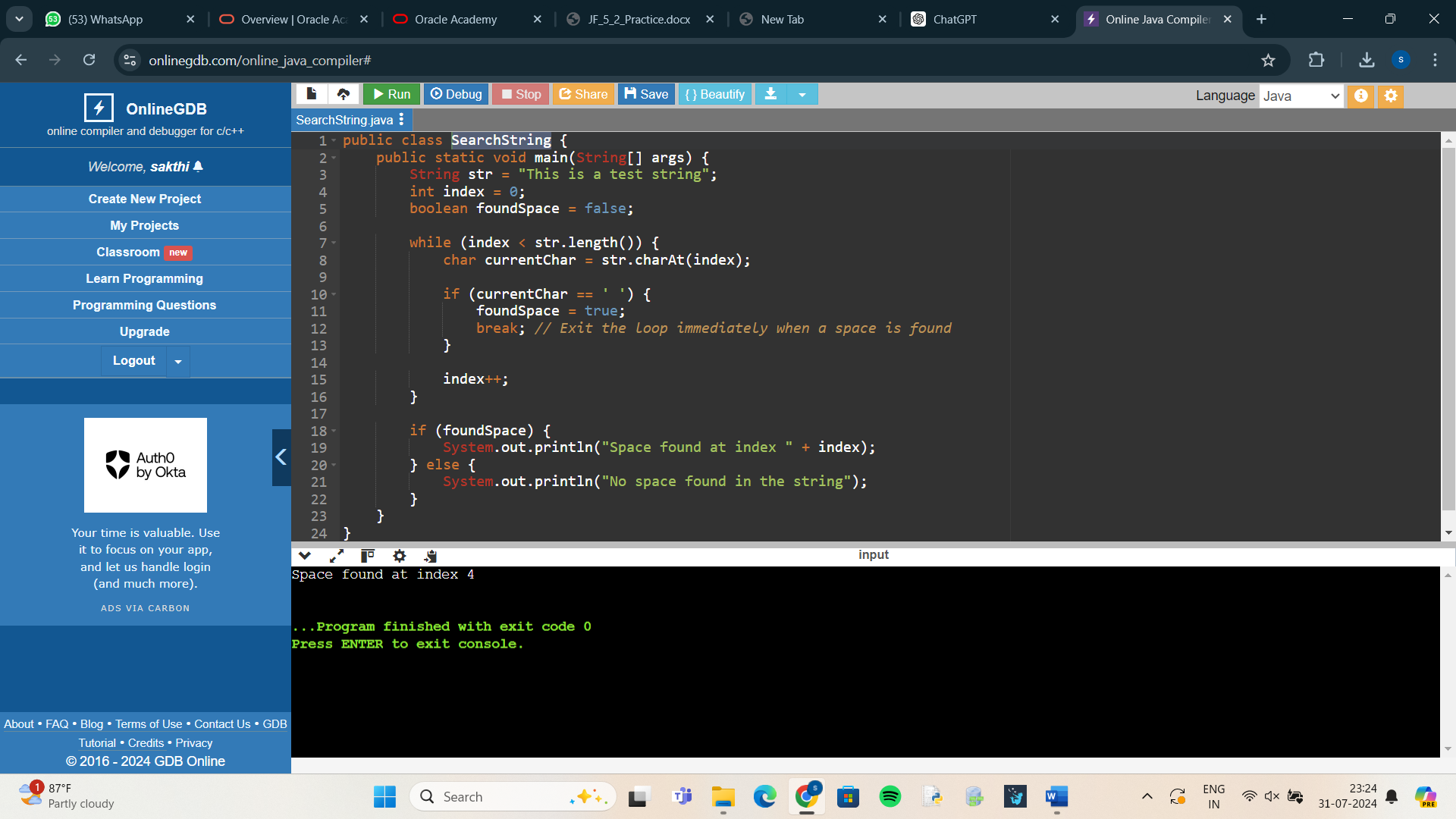
} else {

System.out.println("No space found in the string");

}

}

}



1. 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:**

public class DaysOfWeek {

public static void main(String[] args)

{

String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

int dayOfWeekIndex = 0; // Start with Sunday

// Loop through each day of the year

for (int day = 1; day <= 365; day++) {

System.out.println("Day " + day + ": " + daysOfWeek[dayOfWeekIndex]);

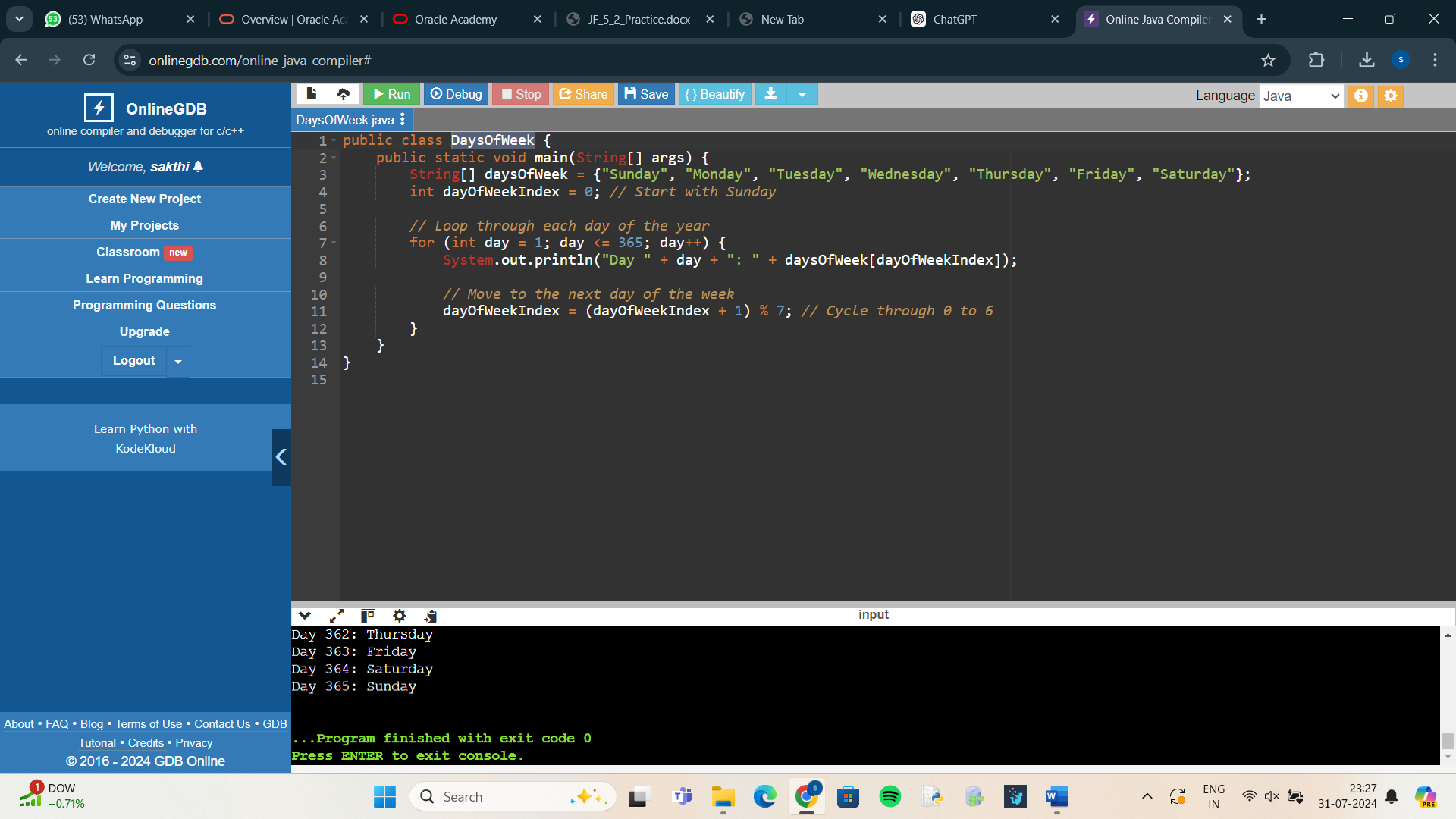
// Move to the next day of the week

dayOfWeekIndex = (dayOfWeekIndex + 1) % 7; // Cycle through 0 to 6

}

}

}



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.

**ANSWER:**

import re

from collections import Counter

def normalize\_string(s):

# Remove punctuation and whitespace, and convert to lowercase

return re.sub(r'[^a-zA-Z]', '', s).lower()

def are\_anagrams(str1, str2):

# Normalize both strings

norm\_str1 = normalize\_string(str1)

norm\_str2 = normalize\_string(str2)

# Compare sorted characters or use Counter to count characters

return Counter(norm\_str1) == Counter(norm\_str2)

# Example usage

string1 = "Listen"

string2 = "Silent!"

if are\_anagrams(string1, string2):

print(f'"{string1}" is an anagram of "{string2}"')

else:

print(f'"{string1}" is not an anagram of "{string2}"')

