

JAVAPROGRAMS

CSA0963

B.MAHESH BABU(192311189)

1. Write a Java method to display the middle character of a string.

Note: a) If the length of the string is odd there will be two middle characters.

b) If the length of the string is even there will be one middle character.

```
public class MiddleCharacter {  
  
    public static void displayMiddleCharacter(String str) {  
        int length = str.length();  
        if (length % 2 == 0) {  
            System.out.println("Middle character: " + str.charAt(length / 2 - 1));  
        } else {  
            System.out.println("Middle characters: " + str.charAt(length / 2 - 1) +  
str.charAt(length / 2));  
        }  
    }  
  
    public static void main(String[] args) {  
        displayMiddleCharacter("example");  
        displayMiddleCharacter("test");  
    }  
}
```

Output

```
java -cp /tmp/prgKl4KVFE/MiddleCharacter  
Middle characters: am  
Middle character: e  
  
=== Code Execution Successful ===
```

2. Write a Java method to check whether a string is a valid password.

Password rules:

A password must have at least ten characters.

A password consists of only letters and digits.

A password must contain at least two digits.

```
import java.util.Scanner; public
```

```
class PasswordValidator {

    public static boolean isValidPassword(String password) {
        if (password.length() < 10) {
            return false;
        }

        int digitCount = 0;
        for (char c : password.toCharArray()) {
            if (!Character.isLetterOrDigit(c)) {
                return false;
            }
            if (Character.isDigit(c)) {
                digitCount++;
            }
        }

        return digitCount >= 2;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a password: ");
        String password = scanner.nextLine();

        if (isValidPassword(password)) {
            System.out.println("Valid password.");
        } else {
            System.out.println("Invalid password.");
        }

        scanner.close();
    }
}
```

Output

```
java -cp /tmp/ppAwrogo5S/PasswordValidato
Enter a password: 123USER123
Valid password.

=== Code Execution Successful ===
```

3. Write a Java recursive method to check if a given array is sorted in ascending order.

```
public class ArrayUtil {
    // Recursive method to check if the array is sorted in ascending order
    public static boolean isSorted(int[] array, int index) { //
        // Base case: If index is at the last element, return true
        if (index >= array.length - 1) {
            return true;
        }
        // Check if the current element is greater than the next element
        if (array[index] > array[index + 1]) { return false;
        }
        // Recursive case: Check the next part of the array
        return isSorted(array, index + 1);
    }

    public static void main(String[] args) {
        int[] sortedArray = {1, 2, 3, 4, 5};
        int[] unsortedArray = {1, 3, 2, 4, 5};

        // Check if the arrays are sorted
        System.out.println("Is the sortedArray sorted? " + isSorted(sortedArray, 0));
        System.out.println("Is the unsortedArray sorted? " + isSorted(unsortedArray, 0));
    }
}
```

OUTPUT

```
Is the sortedArray sorted? true
Is the unsortedArray sorted? false

=== Code Execution Successful ===
```

4. Write a Java program to create a class called "Initializer" with a static block that initializes a static variable 'initialValue' to 1000. Print the value of 'initialValue' before and after creating an instance of "Initializer".

```
4]public class Initializer
{ // Static variable
  static int initialValue;

  // Static block to initialize the static variable
  static {
    initialValue = 1000;
    System.out.println("Static block executed: initialValue = " + initialValue);
  }

  // Constructor
  public Initializer() {
    System.out.println(
      "Constructor
      executed");
  }

  public static void main(String[] args) {
    // Print the value of initialValue before creating an instance
    System.out.println("Value of initialValue before creating an instance: " +
      Initializer.initialValue);

    // Create an instance of Initializer
    Initializer obj = new Initializer();

    // Print the value of initialValue after creating an instance
    System.out.println("Value of initialValue after creating an instance: " +
      Initializer.initialValue);
  }
}
```

OUTPUT

```
Static block executed: initialValue = 1000
Value of initialValue before creating an instance: 1000
Constructor executed
Value of initialValue after creating an instance: 1000

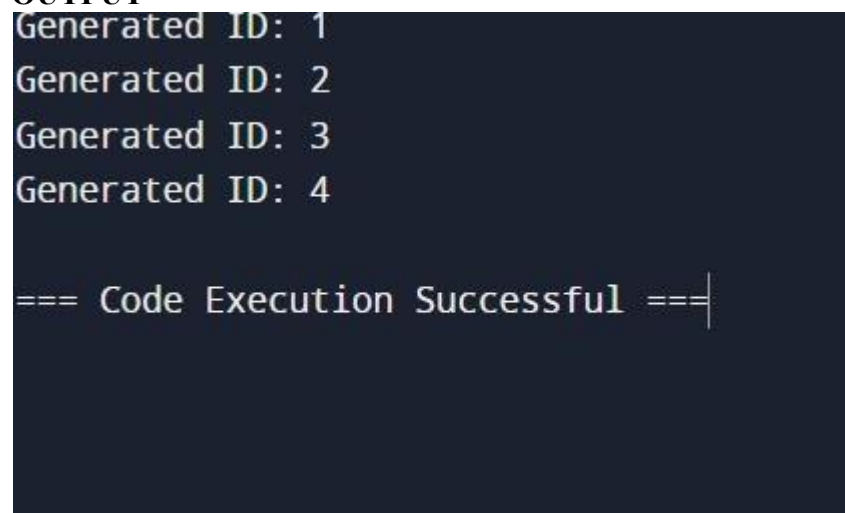
=== Code Execution Successful ===
```

5. Write a Java program to create a class called "IDGenerator" with a static variable 'nextID' and a static method "generateID()" that returns the next ID

and increments 'nextID'. Demonstrate the usage of generateID in the main

```
method. public class IDGenerator {  
    // Static variable to keep track of the next ID  
    private static int nextID = 1;  
  
    // Static method to generate and return the next ID, and increment nextID  
    public static int generateID() {  
        return nextID++;  
    }  
  
    public static void main(String[] args) {  
        // Generating and printing several IDs  
        System.out.println("Generated ID: " + IDGenerator.generateID()); // Output:  
Generated ID: 1  
        System.out.println("Generated ID: " + IDGenerator.generateID()); // Output:  
Generated ID: 2  
        System.out.println("Generated ID: " + IDGenerator.generateID()); // Output:  
Generated ID: 3  
        System.out.println("Generated ID: " + IDGenerator.generateID()); // Output:  
Generated ID: 4  
    }  
}
```

OUTPUT



```
Generated ID: 1  
Generated ID: 2  
Generated ID: 3  
Generated ID: 4  
  
=== Code Execution Successful ===|
```

6. Write a Java program to create a class called Dog with instance variables name and color. Implement a parameterized constructor that takes name and color as parameters and initializes the instance variables. Print the values of the variables.

```
public class Dog {  
    // Instance variables  
    private String name;  
    private String color;
```

```

// Parameterized constructor public
Dog(String name, String color) {
    this.name = name;
    this.color = color;
}

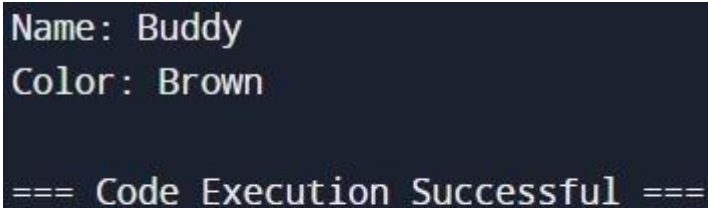
// Method to print the values of the instance variables
public void printDogDetails() {
    System.out.println("Name: " + name);
    System.out.println("Color: " + color);
}

public static void main(String[] args) {
    // Creating an object using the parameterized constructor
    Dog myDog = new Dog("Buddy", "Brown");

    // Printing the details of the dog
    myDog.printDogDetails();
}
}

```

OUTPUT



```

Name: Buddy
Color: Brown

=== Code Execution Successful ===

```

7. Write a Java program to create a class called "Book" with instance variables title, author, and price. Implement a default constructor and two parameterized constructors:

One constructor takes title and author as parameters.

The other constructor takes title, author, and price as parameters.

Print the values of the variables for each constructor.

```

public class Book { // Instance variables private String
title; private String author; private double price;

```

```

// Default constructor
public Book() {
    this.title = "Unknown";
    this.author = "Unknown";
    this.price = 0.0;
}

```

```

    }

    // Parameterized constructor with title and author
    public Book(String title, String author)
    { this.title = title; this.author = author;
      this.price = 0.0; // Default price
    }

    // Parameterized constructor with title, author, and price
    public Book(String title, String author, double price) {
        this.title = title; this.author = author; this.price =
        price;
    }

    // Method to print the values of the instance variables
    public void printBookDetails() { System.out.println("Title:
    " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
        System.out.println();
    }

    public static void main(String[] args) {
        // Creating objects using different constructors
        Book book1 = new Book(); // Default constructor
        Book book2 = new Book("1984", "George Orwell"); // Constructor with title and
author
        Book book3 = new Book("To Kill a Mockingbird", "Harper Lee", 15.99); //
Constructor with title, author, and price

        // Printing the details of each book
        System.out.println("Book 1:");
        book1.printBookDetails();

        System.out.println("Book 2:");
        book2.printBookDetails();

        System.out.println("Book 3:");
        book3.printBookDetails();
    }
}

```

OUTPUT

```
Book 1:
Title: Unknown
Author: Unknown
Price: $0.0

Book 2:
Title: 1984
Author: George Orwell
Price: $0.0

Book 3:
Title: To Kill a Mockingbird
Author: Harper Lee
Price: $15.99

=== Code Execution Successful ===
```

8. Write a Java program to create a class called BankAccount with private instance variables accountNumber and balance. Provide public getter and setter methods to access and modify these variables.

```
public class BankAccount { //
Private instance variables private String accountNumber; private double balance;
```

```
// Constructor to initialize the account
public BankAccount(String accountNumber, double initialBalance) {
    this.accountNumber = accountNumber;
    this.balance = initialBalance;
}

// Getter for accountNumber public
String getAccountNumber() {
    return accountNumber;
}

// Setter for accountNumber
public void setAccountNumber(String accountNumber) {
    this.accountNumber = accountNumber;
}
```



```

// Getter for balance public
double getBalance() {
    return balance;
}

// Setter for balance
public void setBalance(double balance) {
    if (balance >= 0) {
        this.balance = balance;
    } else {
        System.out.println("Balance cannot be negative.");
    }
}

// Main method to test the BankAccount class
public static void main(String[] args) {
    // Create a BankAccount object
    BankAccount account = new BankAccount("123456789", 1000.0);

    // Print initial details
    System.out.println("Account Number: " + account.getAccountNumber());
    System.out.println("Balance: " + account.getBalance());

    // Modify account details
    account.setAccountNumber("987654321");
    account.setBalance(1500.0);

    // Print updated details
    System.out.println("Updated Account Number: " +
account.getAccountNumber());
    System.out.println("Updated Balance: " + account.getBalance());

    // Attempt to set a negative balance
    account.setBalance(-500.0); // This should trigger the error message }
}

```

OUTPUT

```
Account Number: 123456789
Balance: 1000.0
Updated Account Number: 987654321
Updated Balance: 1500.0
Balance cannot be negative.

=== Code Execution Successful ===
```

9. Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.

```
// Define the Playable interface
interface Playable {
    void play();
}

// Implement the Playable interface in the Football class
class Football implements Playable {
    @Override public
    void play() {
        System.out.println("Playing Football");
    }
}

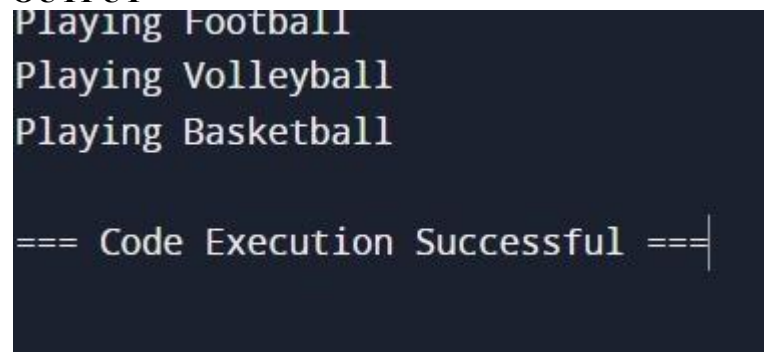
// Implement the Playable interface in the Volleyball class
class Volleyball implements Playable {
    @Override public
    void play() {
        System.out.println("Playing Volleyball");
    }
}

// Implement the Playable interface in the Basketball class
class Basketball implements Playable {
    @Override public
    void play() {
        System.out.println("Playing Basketball");
    }
}
```

```
// Main class to test the implementation
public class SportsTest {
    public static void main(String[] args) {
        // Create instances of each sport
        Playable football = new Football();
        Playable volleyball = new Volleyball();
        Playable basketball = new Basketball();

        // Call the play method for each sport
        football.play();
        volleyball.play();
        basketball.play();
    }
}
```

OUTPUT



```
Playing Football
Playing Volleyball
Playing Basketball

=== Code Execution Successful ===|
```

10. Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd. public class OddNumberCheck {

```
    public static void main(String[] args) {
        try {
            // Test the method with various numbers
            checkEven(4); // Even number
            checkEven(7); // Odd number
            checkEven(10); // Even number
        } catch (OddNumberException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to check if the number is even; throws exception if odd
    public static void checkEven(int number) throws OddNumberException {
        if (number % 2 != 0) {
            throw new OddNumberException("The number " + number + " is odd.");
        } else {

```

```

        System.out.println("The number " + number + " is even.");
    }
}

// Custom exception class for odd numbers class
OddNumberException extends Exception {
    public OddNumberException(String message) {
        super(message);
    }
}

```

OUTPUT

```

The number 4 is even.
The number 7 is odd.

=== Code Execution Successful ===

```

11. Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

```

public class VowelCheck {

    public static void main(String[] args) {
        try {
            // Test the method with various strings
            checkVowels("Hello"); // Contains vowels
            checkVowels("Sky"); // Does not contain vowels
            checkVowels("Rhythm"); // Does not contain vowels
        } catch (NoVowelException e) {
            System.out.println(e.getMessage());
        }
    }

    // Method to check if the string contains vowels
    public static void checkVowels(String input) throws NoVowelException {
        // Regular expression to match vowels (case insensitive)
        if (!input.matches("[aeiouAEIOU].")) {
            throw new NoVowelException("The string \"" + input + "\" does not contain any vowels.");
        } else {
            System.out.println("The string \"" + input + "\" contains vowels.");
        }
    }
}

```

```

    }
}

// Custom exception class
class NoVowelException extends Exception {
    public NoVowelException(String message) {
        super(message);
    }
}

```

OUTPUT

```

The string "Hello" contains vowels.
The string "Sky" does not contain any vowels.

=== Code Execution Successful ===

```

12. Write a Java program to print the following grid. Expected Output :

```

-----
-----
-----
-----
-----
-----
-----
-----
-----
-----

```

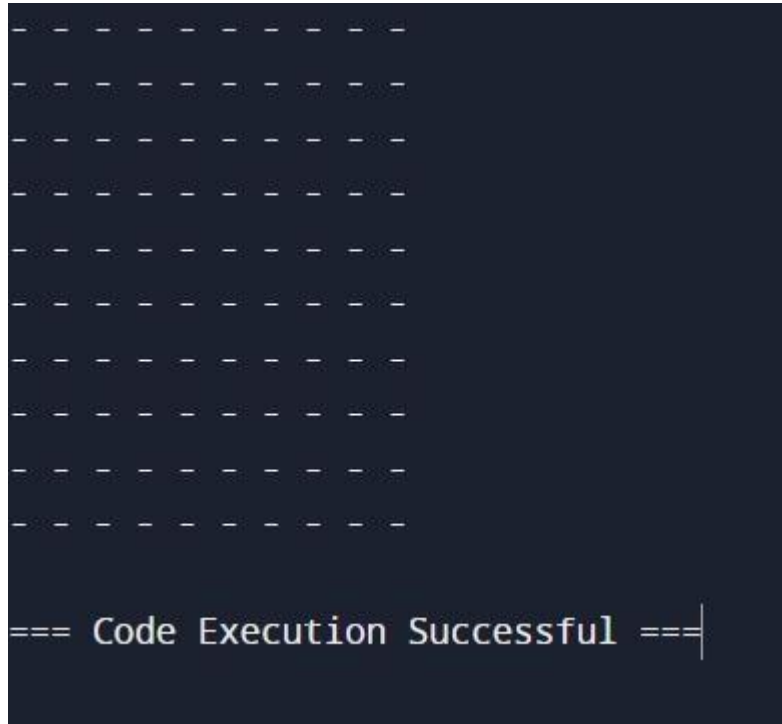
```

public class PrintGrid {
    public static void main(String[] args) {
        // Define the number of rows and columns
        int rows = 10; int columns = 10;

        // Loop through each row for
        (int i = 0; i < rows; i++) {
            // Loop through each column in the current row
            for (int j = 0; j < columns; j++) {
                // Print a dash and a space
                System.out.print("- ");
            }
            // Move to the next line after each row
            System.out.println();
        }
    }
}

```

OUTPUT

A screenshot of a code execution environment with a dark background. At the bottom, it displays the message "=== Code Execution Successful ===" in a light green monospace font, followed by a vertical cursor line. Above this message, there are several horizontal dashed lines, suggesting that some output was printed but is not visible in the image.

13. Write a Java program to create a generic method that takes two lists of the same type and merges them into a single list. This method alternates the elements of each list.

```
import java.util.ArrayList;
import java.util.List;
```

```
public class MergeLists {
```

```
    public static void main(String[] args) {
```

```
        // Create two sample lists
```

```
        List<Integer> list1 = new ArrayList<>();
```

```
        List<Integer> list2 = new ArrayList<>();
```

```
        // Add elements to the first list
```

```
        list1.add(1);
```

```
        list1.add(3);
```

```
        list1.add(5);
```

```
        // Add elements to the second list
```

```
        list2.add(2);
```

```
        list2.add(4);
```

```
        list2.add(6);
```

```
        // Call the generic merge method
```

```
        List<Integer> mergedList = mergeListsAlternating(list1, list2);
```

```
        // Print the merged list
```

```

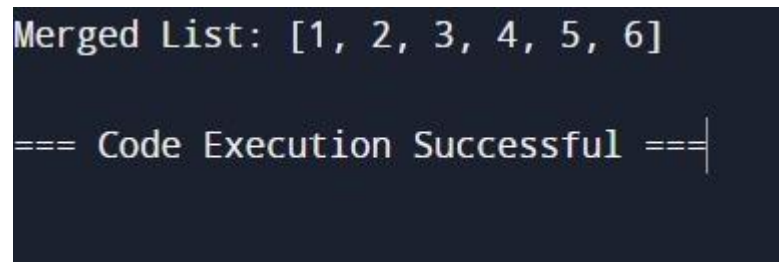
        System.out.println("Merged List: " + mergedList);
    }
    // Generic method to merge two lists of the same type
    public static <T> List<T> mergeListsAlternating(List<T> list1, List<T> list2) {
        List<T> mergedList = new ArrayList<>();
        int size1 = list1.size(); int size2 =
        list2.size(); int maxSize =
        Math.max(size1, size2);

        // Alternating elements from both lists
        for (int i = 0; i < maxSize; i++) { if (i
        < size1) {
            mergedList.add(list1.get(i));
        } if (i <
        size2) {
            mergedList.add(list2.get(i));
        }
        }

        return mergedList;
    }
}

```

OUTPUT



```

Merged List: [1, 2, 3, 4, 5, 6]

=== Code Execution Successful ===

```

14. Write a Java program to sort an array of given integers using the Selection Sort Algorithm

```

import java.util.Scanner;

public class SelectionSortExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input array size
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        // Initialize the array
        int[] array = new int[n];

        // Input array elements
        System.out.println("Enter the elements of the array:");
    }
}

```

```

    for (int i = 0; i < n; i++) {
        array[i] = scanner.nextInt();
    }
    // Perform Selection Sort
    selectionSort(array);

    // Display the sorted array
    System.out.println("Sorted array:");
    for (int element : array) {
        System.out.print(element + " ");
    }

    scanner.close();
}

// Selection Sort function
public static void selectionSort(int[] array) {
    int n = array.length;

    // Traverse through all array elements
    for (int i = 0; i < n - 1; i++) {
        // Find the minimum element in the unsorted
        // part int minIndex = i; for (int j = i + 1; j < n;
        // j++) {
        //     if (array[j] < array[minIndex]) {
        //         minIndex = j;
        //     }
        // }

        // Swap the found minimum element with the first element
        int temp = array[minIndex];
        array[minIndex] = array[i];
        array[i] = temp;
    }
}
}

```

OUTPUT

```

Enter the number of elements in the array: 5
Enter the elements of the array:
6 7 8 9 6
Sorted array:
6 6 7 8 9
=== Code Execution Successful ===

```


15. Write a Java program to find a specified element in a given array of elements using Binary Search.

```
import java.util.Arrays;
import java.util.Scanner;

public class BinarySearchExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input array size
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();

        // Initialize the array
        int[] array = new int[n];

        // Input array elements
        System.out.println("Enter the elements of the array (sorted order):");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }

        // Input the element to search for
        System.out.print("Enter the element to search for: ");
        int key = scanner.nextInt();

        // Perform Binary Search int result =
        binarySearch(array, key);

        // Display the result
        if (result == -1) {
            System.out.println("Element not found in the array.");
        } else {
            System.out.println("Element found at index: " + result);
        }

        scanner.close();
    }

    // Binary Search function
    public static int binarySearch(int[] array, int key) {
        int left = 0; int right =
        array.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
```

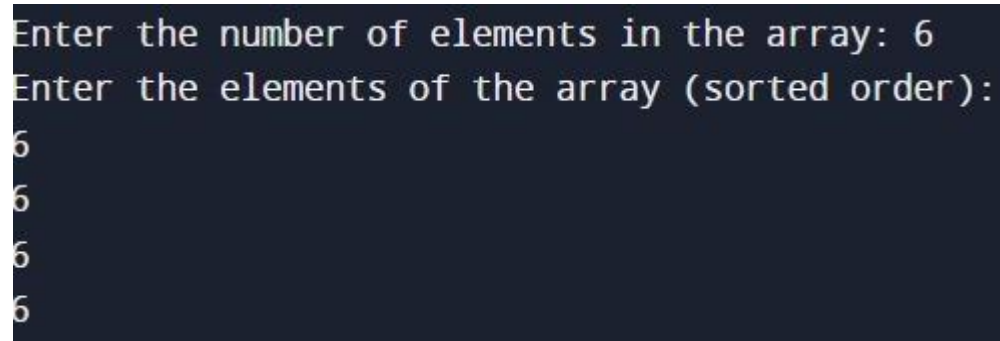
```

        // Check if the key is present at mid
        if (array[mid] == key) {
            return mid; // Element found, return index
        }
        // If the key is greater, ignore the left half
        if (array[mid] < key) {
            left = mid + 1;
        }
        // If the key is smaller, ignore the right half
        else {
            right = mid - 1;
        }
    }

    // Element not found
    return -1;
}
}

```

OUTPUT



```

Enter the number of elements in the array: 6
Enter the elements of the array (sorted order):
6
6
6
6
6

```

16. Write a Java program to find sequences of lowercase letters joined by an underscore.

```

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.Scanner;

public class LowercaseUnderscoreMatcher {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Regular expression to match sequences of lowercase letters joined by an underscore
        String regex = "\\b[a-z]+_[a-z]+\\b";

        // Compile the regular expression
        Pattern pattern = Pattern.compile(regex);
    }
}

```

```

System.out.println("Enter a sentence:");
String input = scanner.nextLine();

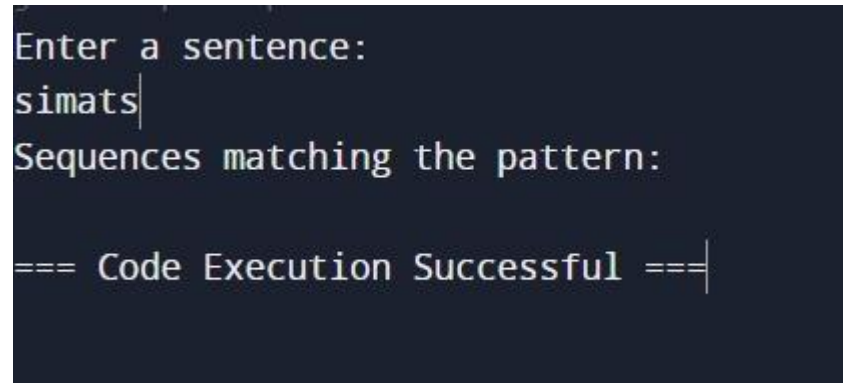
// Match the pattern in the input sentence
Matcher matcher = pattern.matcher(input);

System.out.println("Sequences matching the pattern:");
while (matcher.find()) {
    System.out.println(matcher.group());
}

scanner.close();
}
}

```

OUTPUT



```

Enter a sentence:
simats
Sequences matching the pattern:

=== Code Execution Successful ===

```

17. Write a Java program that matches a word containing 'g', not at the start or end of the word.

```

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.Scanner;

public class WordMatcher {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Regular expression to match a word containing 'g' but not at the start or end
        String regex = "\\b[a-fh-zA-FH-Z]g[a-zA-Z]\\b";

        // Compile the regular expression
        Pattern pattern = Pattern.compile(regex);

        System.out.println("Enter a sentence:");
        String input = scanner.nextLine();

        // Match the pattern in the input sentence
        Matcher matcher = pattern.matcher(input);
    }
}

```

```
        System.out.println("Words matching the pattern:");
        while (matcher.find()) {
            System.out.println(matcher.group());
        }

        scanner.close();
    }
}
```

OUTPUT

```
Enter a sentence:
saveetha
Words matching the pattern:

=== Code Execution Successful ===
```