

Subject: Algorithm and Data Structure

Assignment 1

Solve the assignment with the following thing to be added in each question.

- Program
- Flow chart
- Explanation
- Output
- Time and Space complexity

1. Armstrong Number

Problem: Write a Java program to check if a given number is an Armstrong number.

Test Cases:

Input:

153

Output: true

Input: 123

Output: false

Solution:

-Program

```
import java.util.Scanner;

public class ArmstrongNumber {
    public static boolean isArmstrong(int number){
        int originalNumber = number, result = 0, digits = 0;

        while (originalNumber != 0){
            originalNumber /= 10;
            digits++;
        }
        originalNumber = number;

        while(originalNumber != 0){
            int remainder = originalNumber % 10;
            result += Math.pow(remainder, digits);
            originalNumber /= 10;
        }
        return result == number;
    }
}
```

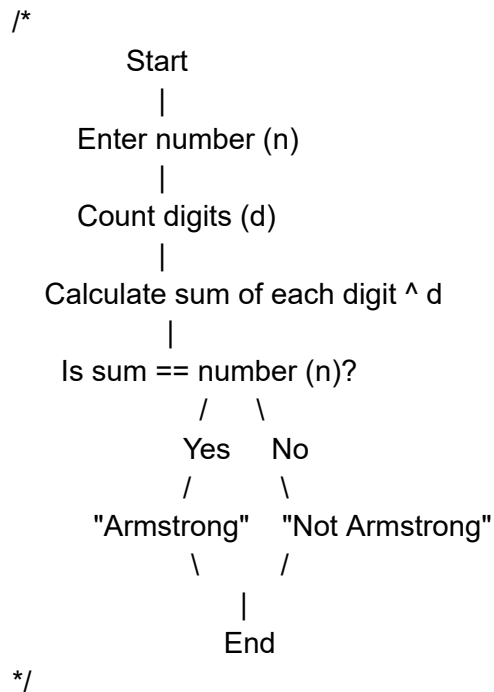
```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter a number: ");
    int number = scanner.nextInt();

    if(isArmstrong(number)){
        System.out.println("True");
    }else{
        System.out.println("False");
    }
    scanner.close();
}
}

```

-Flow Chart



-Explanation

The program first counts the number of digits in the input number.

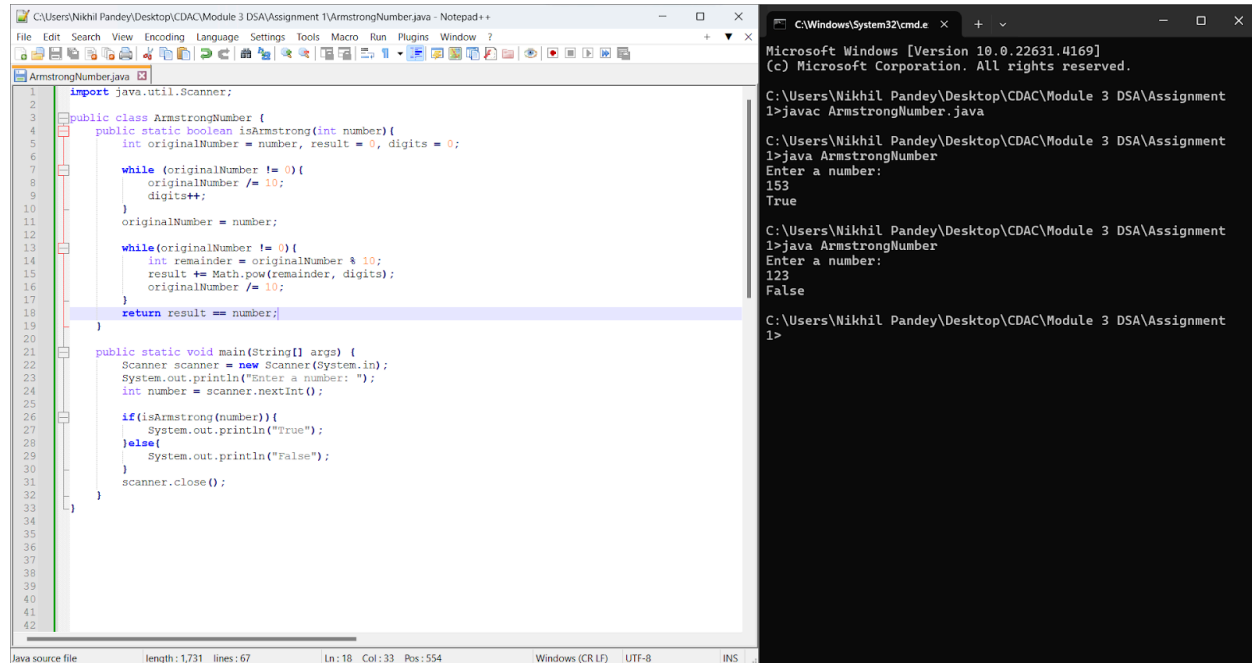
Then it calculates the sum of each digit raised to the power of the total number of digits.

Finally, it checks if this calculated sum is equal to the original number. If yes, the number is an Armstrong number; otherwise, it is not.

Armstrong Number Definition: A number is called an Armstrong number if the sum of its digits raised to the power of the number of digits equals the number itself. For example, 153 is an Armstrong number because

$1^3 + 5^3 + 3^3 = 153$.

-Output



The image shows a Java source file named `ArmstrongNumber.java` in a Notepad++ window and its execution in a Windows Command Prompt.

Java Source File:

```
1 import java.util.Scanner;
2
3 public class ArmstrongNumber {
4     public static boolean isArmstrong(int number){
5         int originalNumber = number, result = 0, digits = 0;
6
7         while (originalNumber != 0){
8             originalNumber /= 10;
9             digits++;
10        }
11        originalNumber = number;
12
13        while(originalNumber != 0){
14            int remainder = originalNumber % 10;
15            result += Math.pow(remainder, digits);
16            originalNumber /= 10;
17        }
18        return result == number;
19    }
20
21    public static void main(String[] args) {
22        Scanner scanner = new Scanner(System.in);
23        System.out.println("Enter a number: ");
24        int number = scanner.nextInt();
25
26        if(isArmstrong(number)){
27            System.out.println("True");
28        }else{
29            System.out.println("False");
30        }
31        scanner.close();
32    }
33 }
34
35
36
37
38
39
40
41
42
```

Command Prompt Output:

```
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>javac ArmstrongNumber.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java ArmstrongNumber
Enter a number:
153
True

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java ArmstrongNumber
Enter a number:
123
False

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>
```

-Time and Space complexity

Time Complexity:

The time complexity of this program is $O(d)$, where d is the number of digits in the input number. This is because we iterate over each digit twice: once for counting the digits and once for calculating the sum of powers.

Space Complexity:

The space complexity of this program is $O(1)$ since the space used by the program does not depend on the size of the input, aside from a few integer variables.

2. Prime Number

Problem: Write a Java program to check if a given number is prime.

Test Cases:

Input: 29

Output: true

Input: 15

Output: false

Solution:

-Program

```
import java.util.Scanner;

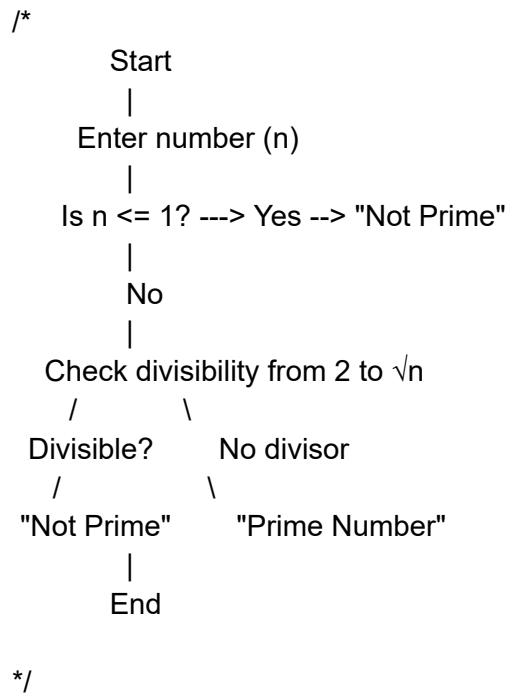
public class PrimeNumber {
    public static boolean isPrime(int number) {
        if(number <= 1){
            return false;
        }
        for(int i = 2; i < Math.sqrt(number); i++) {
            if(number % i == 0){
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int number = scanner.nextInt();

        if(isPrime(number)){
            System.out.println("True");
        }else{
            System.out.println("False");
        }

        scanner.close();
    }
}
```

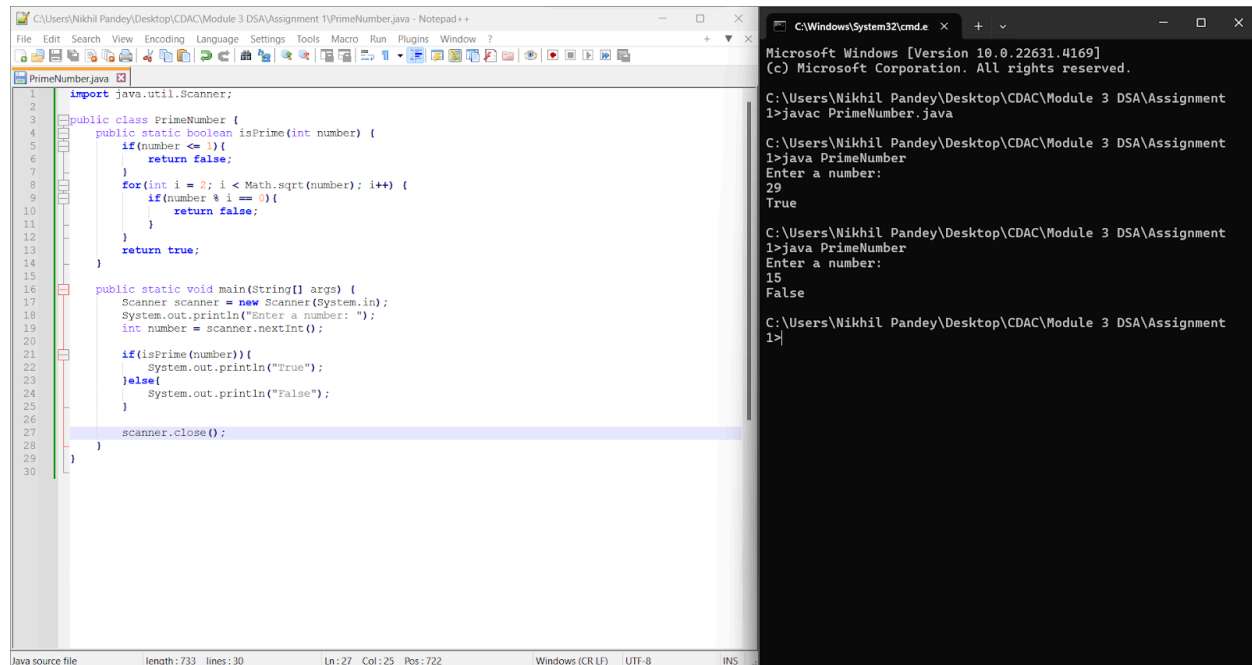
-Flow Chart



-Explanation

- A prime number is a natural number greater than 1 that is not divisible by any number other than 1 and itself.
- The program checks if the input number is less than or equal to 1, in which case it is not a prime number.
- If the number is greater than 1, the program checks divisibility by numbers from 2 up to the square root of the number. If it finds a divisor, the number is not prime.
- If no divisors are found, the number is prime.

-Output



The screenshot displays two windows side-by-side. The left window is a Notepad++ editor showing the source code of a Java program named PrimeNumber.java. The code defines a static method isPrime that checks if a number is prime by testing divisibility up to its square root. The main method uses a Scanner to take user input and prints the result. The right window is a Windows Command Prompt showing the compilation and execution of the program. It shows the command 'javac PrimeNumber.java' being executed successfully, followed by two runs of 'java PrimeNumber'. The first run takes input '29' and outputs 'True', while the second run takes input '15' and outputs 'False'.

```
import java.util.Scanner;

public class PrimeNumber {
    public static boolean isPrime(int number) {
        if(number <= 1){
            return false;
        }
        for(int i = 2; i < Math.sqrt(number); i++) {
            if(number % i == 0){
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int number = scanner.nextInt();

        if(isPrime(number)){
            System.out.println("True");
        }else{
            System.out.println("False");
        }

        scanner.close();
    }
}
```

```
Microsoft Windows [Version 10.0.22621.4169]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>javac PrimeNumber.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java PrimeNumber
Enter a number:
29
True

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java PrimeNumber
Enter a number:
15
False

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>
```

-Time and Space complexity

Time Complexity:

- The time complexity is $O(\sqrt{n})$ because we only check divisibility up to the square root of the number.

Space Complexity:

- The space complexity is $O(1)$ as the program only uses a few integer variables.

3. Factorial

Problem: Write a Java program to compute the factorial of a given number.

Test Cases:

Input: 5

Output: 120

Input: 0

Output: 1

Solution:

-Program

```
import java.util.Scanner;

public class Factorial {
    public static long factorial(int number) {
        long result = 1;
        for(int i = 2; i <= number; i++) {
            result *= i;
        }
        return result;
    }

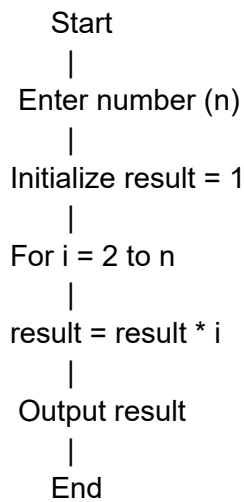
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number");
        int number = scanner.nextInt();

        long fact = factorial(number);
        System.out.println(fact);

        scanner.close();
    }
}
```

-Flow Chart

/*



*/

-Explanation

Factorial of a number n is the product of all positive integers less than or equal to n . Mathematically, it is denoted as $n!$ and is computed as $n! = n \times (n-1) \times (n-2) \times \dots \times 1$.

The program initializes result to 1, then multiplies it by each integer from 2 to the input number. If the input number is 0 or 1, the factorial is 1 (by definition).

-Output

The screenshot displays two windows. The left window is a Notepad++ editor showing the source code of a Java program named 'Factorial.java'. The code imports 'java.util.Scanner', defines a 'Factorial' class with a static method 'factorial' that calculates the factorial using a loop, and a 'main' method that uses a scanner to take user input and print the result. The right window is a Windows Command Prompt showing the execution of the program. It first compiles the code with 'javac Factorial.java' and then runs it with 'java Factorial'. The program prompts 'Enter a number' and shows two examples: input 5 resulting in output 120, and input 0 resulting in output 1.

```
1 import java.util.Scanner;
2
3 public class Factorial {
4     public static long factorial(int number) {
5         long result = 1;
6         for(int i = 2; i <= number; i++) {
7             result *= i;
8         }
9         return result;
10    }
11
12
13    public static void main(String[] args) {
14        Scanner scanner = new Scanner(System.in);
15        System.out.println("Enter a number");
16        int number = scanner.nextInt();
17
18        long fact = factorial(number);
19        System.out.println(fact);
20
21        scanner.close();
22    }
23 }
```

```
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>javac Factorial.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java Factorial
Enter a number
5
120

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>java Factorial
Enter a number
0
1

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment
1>
```

Java source file length: 543 lines: 23 Ln: 23 Col: 2 Pos: 544 Windows (CR LF) UTF-8 INS

-Time and Space complexity

Time Complexity:

- The time complexity is **$O(n)$** , where n is the input number, since we loop from 2 to the number to calculate its factorial.

Space Complexity:

- The space complexity is **$O(1)$** as the program only uses a few variables to store the result and the loop counter.

4. Fibonacci Series

Problem: Write a Java program to print the first n numbers in the Fibonacci series.

Test Cases:

Input: $n = 5$

Output: [0, 1, 1, 2, 3]

Input: $n = 8$

Output: [0, 1, 1, 2, 3, 5, 8, 13]

Solution:

-Program

```
import java.util.Scanner;
import java.util.ArrayList;

public class FibonacciSeries{
    public static ArrayList <Integer> getFibonacciSeries(int n){
        ArrayList<Integer> series = new ArrayList<>();

        if(n <= 0)return series;
        series.add(0);
        if (n == 1) return series;

        int first = 0, second = 1;
        series.add(second);

        for(int i = 2; i < n; i++){
            int next = first + second;
            series.add(next);
            first = second;
            second = next;
        }
        return series;
    }
}
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the value of n: ");
    int n = scanner.nextInt();

    ArrayList<Integer> fibonacciSeries = getFibonacciSeries(n);
    System.out.println("Fibonacci Series: " + fibonacciSeries);

    scanner.close();
}
}

```

-Flow Chart

```

/*
    Start
    |
    Enter value of n
    |
    Is n <= 0? ---> Yes --> Return empty series
    |
    No
    |
    Initialize first = 0, second = 1
    |
    Add first and second to series
    |
    Loop i from 3 to n
    |
    Calculate next = first + second
    |
    Add next to series
    |
    Update first = second, second = next
    |
    End
*/

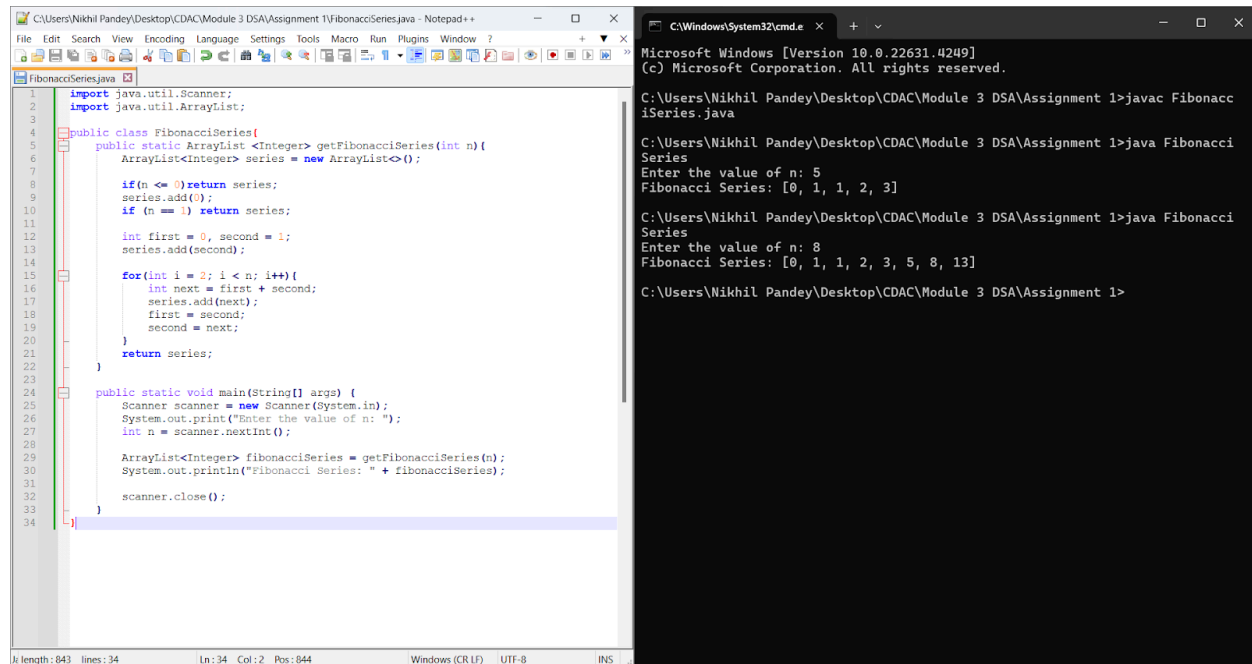
```

-Explanation

Fibonacci Series: A sequence where each number is the sum of the two preceding ones, starting from 0 and 1.

The program initializes the first two Fibonacci numbers (first = 0 and second = 1), adds them to the list, and computes successive numbers by adding the previous two. If n is 0 or less, the program returns an empty list, and if $n = 1$, it returns only the first Fibonacci number. The program iterates until the n th Fibonacci number is generated and printed.

-Output



The screenshot displays two windows. The left window is a Notepad++ editor showing the source code of a Java program named `FibonacciSeries.java`. The code defines a `FibonacciSeries` class with a `getFibonacciSeries(int n)` method that returns an `ArrayList<Integer>`. It handles edge cases for $n \leq 0$ and $n = 1$, then iterates from $i = 2$ to n to calculate the series. The `main` method uses a `Scanner` to take user input and prints the resulting series. The right window is a Windows Command Prompt showing the compilation and execution of the program. It demonstrates two runs: one for $n = 5$ resulting in the series `[0, 1, 1, 2, 3]`, and another for $n = 8$ resulting in the series `[0, 1, 1, 2, 3, 5, 8, 13]`.

```
1 import java.util.Scanner;
2 import java.util.ArrayList;
3
4 public class FibonacciSeries{
5     public static ArrayList<Integer> getFibonacciSeries(int n){
6         ArrayList<Integer> series = new ArrayList<>();
7
8         if(n <= 0) return series;
9         series.add(0);
10        if (n == 1) return series;
11
12        int first = 0, second = 1;
13        series.add(second);
14
15        for(int i = 2; i <= n; i++){
16            int next = first + second;
17            series.add(next);
18            first = second;
19            second = next;
20        }
21        return series;
22    }
23
24    public static void main(String[] args) {
25        Scanner scanner = new Scanner(System.in);
26        System.out.print("Enter the value of n: ");
27        int n = scanner.nextInt();
28
29        ArrayList<Integer> fibonacciSeries = getFibonacciSeries(n);
30        System.out.println("Fibonacci Series: " + fibonacciSeries);
31
32        scanner.close();
33    }
34 }
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac FibonacciSeries.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java FibonacciSeries
Enter the value of n: 5
Fibonacci Series: [0, 1, 1, 2, 3]

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java FibonacciSeries
Enter the value of n: 8
Fibonacci Series: [0, 1, 1, 2, 3, 5, 8, 13]

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity: The time complexity is $O(n)$, where n is the input number. The program iterates n times to calculate each Fibonacci number.

Space Complexity: The space complexity is $O(n)$ because we store the Fibonacci series in a list of size n .

5. Find GCD

Problem: Write a Java program to find the Greatest Common Divisor (GCD) of two numbers.

Test Cases:

Input: a = 54, b = 24

Output: 6

Input: a = 17, b = 13

Output: 1

Solution:

-Program

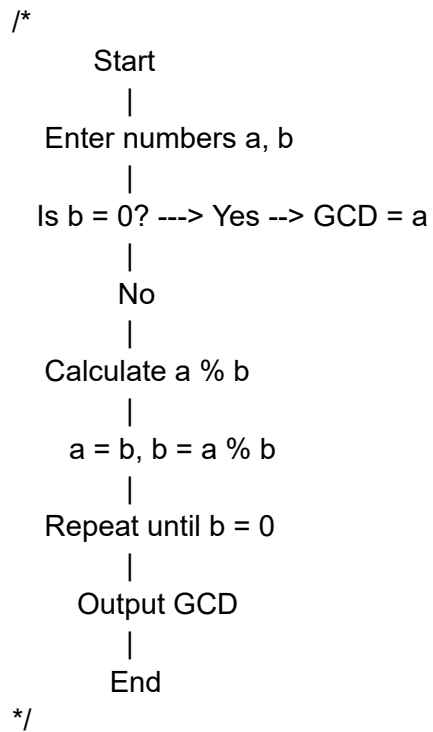
```
import java.util.Scanner;

public class GCD {
    public static int findGCD(int a, int b) {
        if(b == 0) {
            return a;
        }
        return findGCD(b, a % b);
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first number (a): ");
        int a = scanner.nextInt();
        System.out.print("Enter second number (b): ");
        int b = scanner.nextInt();

        int gcd = findGCD(a, b);
        System.out.println("GCD of " + a + " and " + b + " is: " + gcd);

        scanner.close();
    }
}
```

-Flow Chart



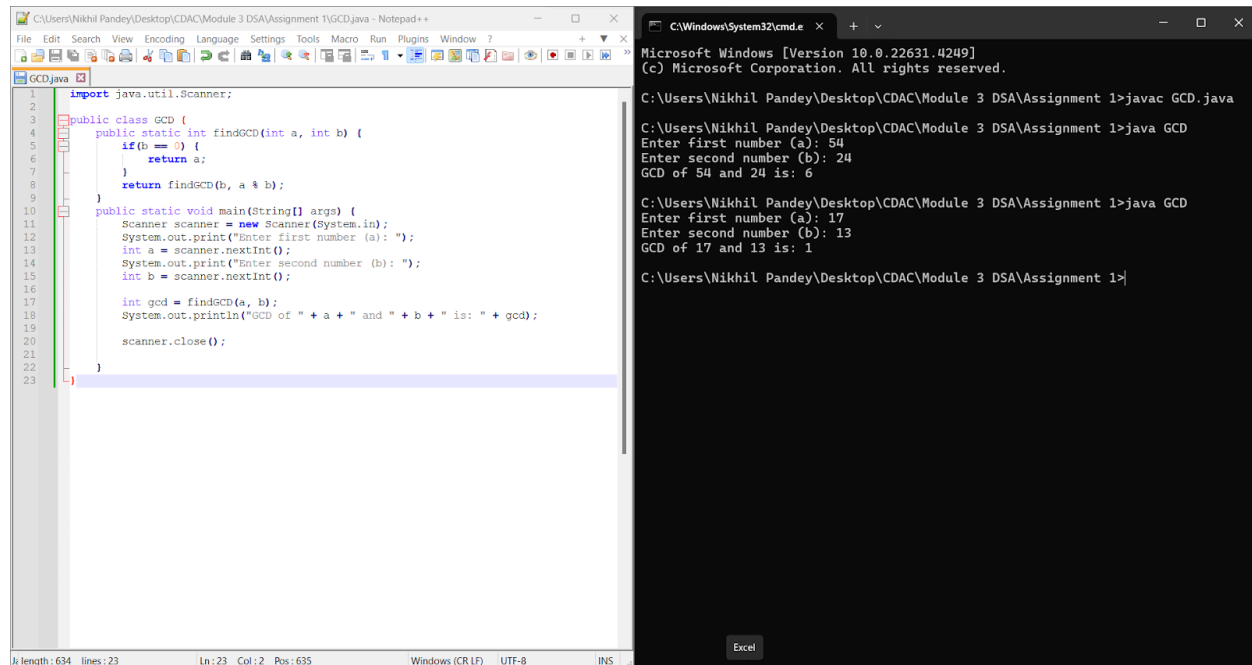
-Explanation

Euclidean Algorithm: The GCD of two numbers a and b is calculated by repeatedly applying the equation $\text{GCD}(a, b) = \text{GCD}(b, a \% b)$ until b becomes zero. At that point, a is the GCD.

The program recursively calls the findGCD() function until the second number b becomes zero, and returns the first number a as the GCD.

The input numbers are taken from the user, and the result is displayed.

-Output



The image shows a Notepad++ editor window on the left and a Windows Command Prompt window on the right. The Notepad++ window displays the source code for a Java program named GCD.java. The code implements the Euclidean algorithm for finding the Greatest Common Divisor (GCD) of two integers. It includes a recursive method findGCD and a main method that takes user input for two numbers and prints the result.

```
1 import java.util.Scanner;
2
3 public class GCD {
4     public static int findGCD(int a, int b) {
5         if(b == 0) {
6             return a;
7         }
8         return findGCD(b, a % b);
9     }
10
11     public static void main(String[] args) {
12         Scanner scanner = new Scanner(System.in);
13         System.out.print("Enter first number (a): ");
14         int a = scanner.nextInt();
15         System.out.print("Enter second number (b): ");
16         int b = scanner.nextInt();
17
18         int gcd = findGCD(a, b);
19         System.out.println("GCD of " + a + " and " + b + " is: " + gcd);
20
21         scanner.close();
22     }
23 }
```

The Command Prompt window shows the execution of the program. It displays the commands used to compile and run the program, followed by the user input and the output of the program.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac GCD.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java GCD
Enter first number (a): 54
Enter second number (b): 24
GCD of 54 and 24 is: 6

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java GCD
Enter first number (a): 17
Enter second number (b): 13
GCD of 17 and 13 is: 1

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity:

The time complexity is $O(\log(\min(a, b)))$ because with each step, the value of b is reduced by the modulus operation until b becomes zero.

Space Complexity:

The space complexity is $O(\log(\min(a, b)))$ due to the recursive calls in the Euclidean algorithm. If an iterative version is used, the space complexity would be $O(1)$.

6. Find Square Root

Problem: Write a Java program to find the square root of a given number (using integer approximation).

Test Cases:

Input: x = 16

Output: 4

Input: x = 27

Output: 5

Solution:

-Program

```
import java.util.Scanner;

public class SquareRoot {
    public static int findSquareRoot(int x){
        if(x < 0){
            System.out.println("Square root of negative numbers is not supported.");
            return -1;
        }
        if (x == 0 || x == 1) {
            return x;
        }
        int start = 1, end = x, ans = 0;

        while (start <= end){
            int mid = (start + end) / 2;
            if (mid * mid == x) {
                return mid;
            }
            if (mid * mid < x) {
                start = mid + 1;
                ans = mid;
            } else {
                end = mid - 1;
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int x = scanner.nextInt();
    }
}
```

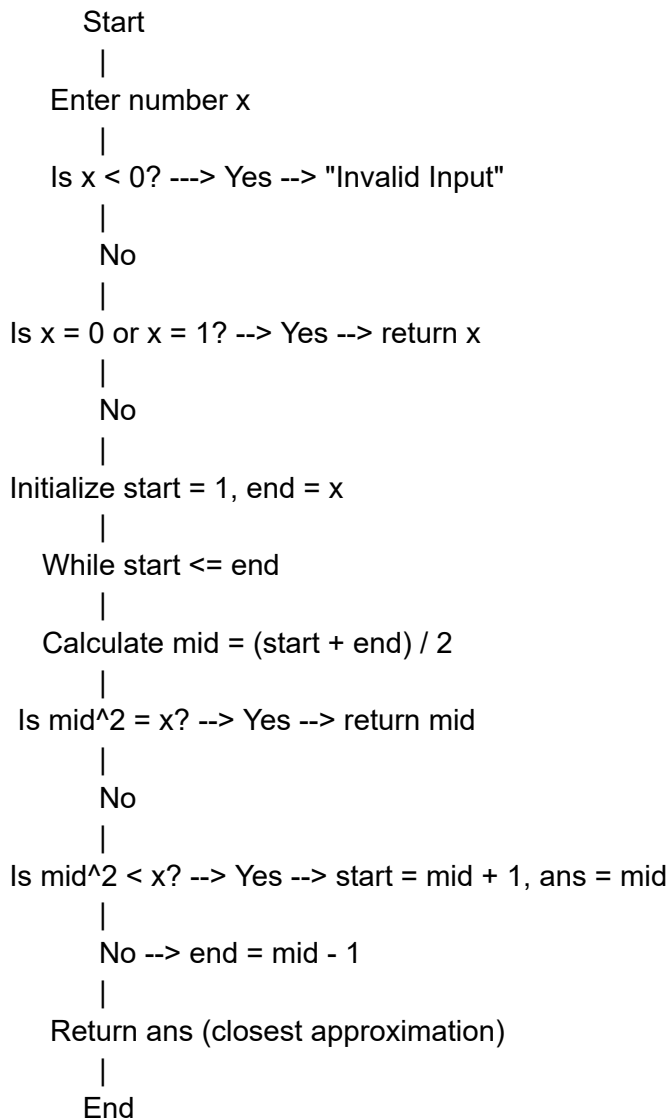
```

    int sqrt = findSquareRoot(x);
    if (sqrt != 1) {
        System.out.println("The integer square root of " + x + " is: " + sqrt);
    }
    scanner.close();
}
}

```

-Flow Chart

/*



*/

-Explanation

The program uses binary search to find the square root. The search space is between 1 and the given number.

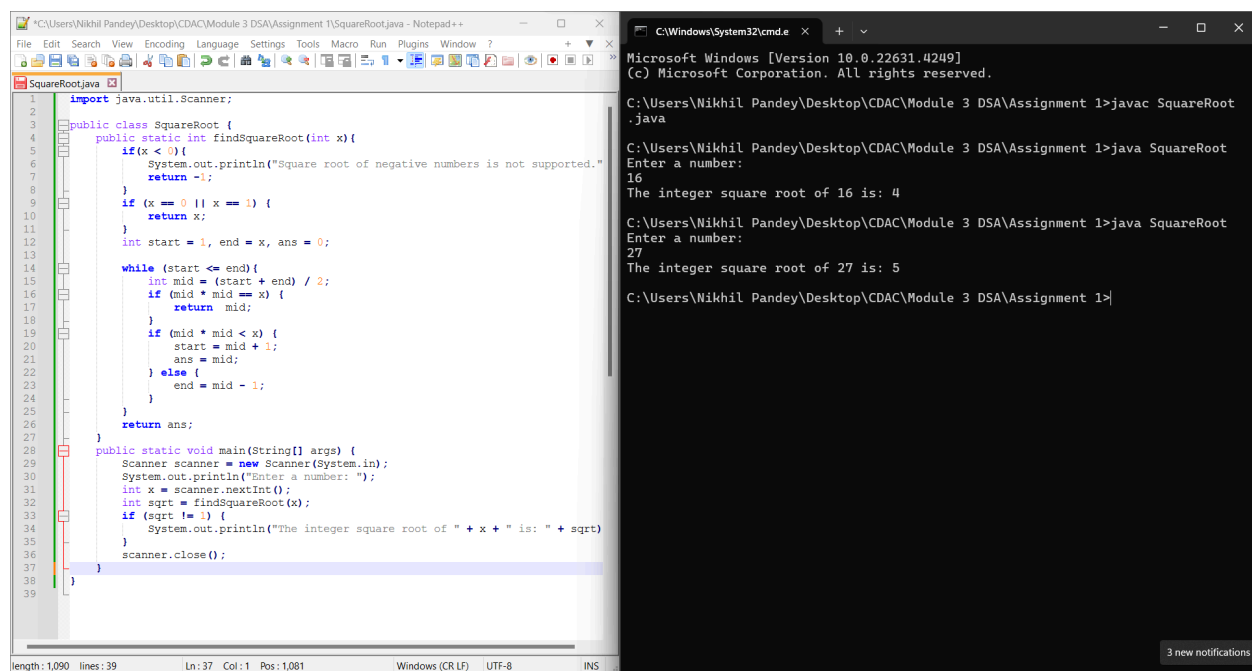
At each step, it checks if the middle number (mid) squared is equal to the given number. If yes, it returns the exact square root.

If $\text{mid} * \text{mid}$ is less than the number, the search continues in the right half of the search space; otherwise, it continues in the left half.

The variable ans keeps track of the closest integer approximation of the square root.

If the input is less than 0, the program returns an error message since the square root of negative numbers is not supported.

-Output



```
1 import java.util.Scanner;
2
3 public class SquareRoot {
4     public static int findSquareRoot(int x){
5         if(x < 0){
6             System.out.println("Square root of negative numbers is not supported.");
7             return -1;
8         }
9         if(x == 0 || x == 1){
10            return x;
11        }
12        int start = 1, end = x, ans = 0;
13
14        while (start <= end){
15            int mid = (start + end) / 2;
16            if (mid * mid == x){
17                return mid;
18            }
19            if (mid * mid < x){
20                start = mid + 1;
21            }
22            else {
23                end = mid - 1;
24            }
25        }
26        return ans;
27    }
28
29    public static void main(String[] args) {
30        Scanner scanner = new Scanner(System.in);
31        System.out.println("Enter a number: ");
32        int x = scanner.nextInt();
33        int sqrt = findSquareRoot(x);
34        if (sqrt != -1) {
35            System.out.println("The integer square root of " + x + " is: " + sqrt);
36        }
37        scanner.close();
38    }
39 }
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac SquareRoot.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java SquareRoot
Enter a number:
16
The integer square root of 16 is: 4

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java SquareRoot
Enter a number:
27
The integer square root of 27 is: 5

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity:

The time complexity is $O(\log(x))$, where x is the input number. This is because we are using binary search to find the square root.

Space Complexity:

The space complexity is $O(1)$ as the program only uses a few variables (start, end, mid, ans) regardless of the size of the input.

7. Find Repeated Characters in a String

Problem: Write a Java program to find all repeated characters in a string.

Test Cases:

Input: "programming"

Output: ['r', 'g', 'm']

Input: "hello"

Output: ['l']

Solution:

-Program

```
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;

public class RepeatedCharacters {
    public static ArrayList<Character> findRepeatedChars(String str){
        HashMap<Character, Integer> charCountMap = new HashMap<>();
        ArrayList<Character> repeatedChars = new ArrayList<>();

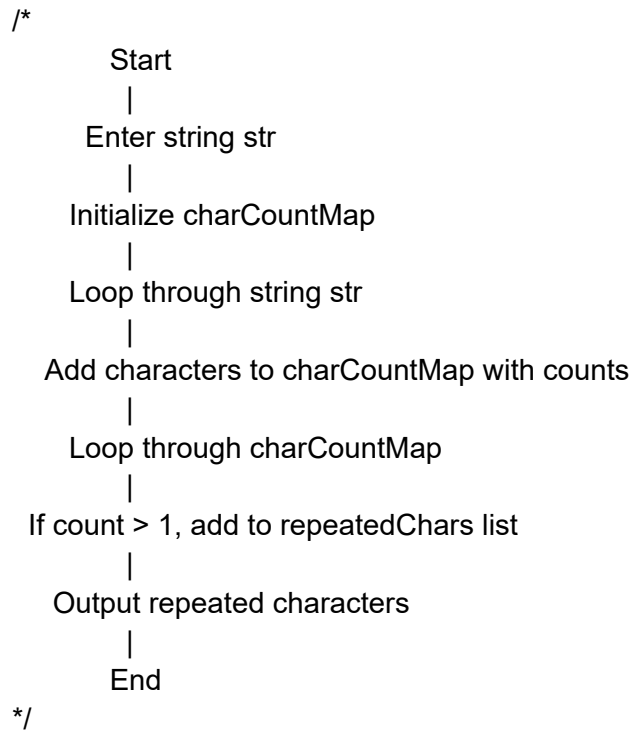
        for(int i = 0; i < str.length(); i++){
            char currentChar = str.charAt(i);
            charCountMap.put(currentChar, charCountMap.getOrDefault(currentChar,0) + 1);
        }
        for(char key : charCountMap.keySet()){
            if (charCountMap.get(key) > 1) {
                repeatedChars.add(key);
            }
        }
        return repeatedChars;
    }

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

        ArrayList<Character> result = findRepeatedChars(input);
        System.out.println("Repeated Characters: " + result);

        scanner.close();
    }
}
```

-Flow Chart



-Explanation

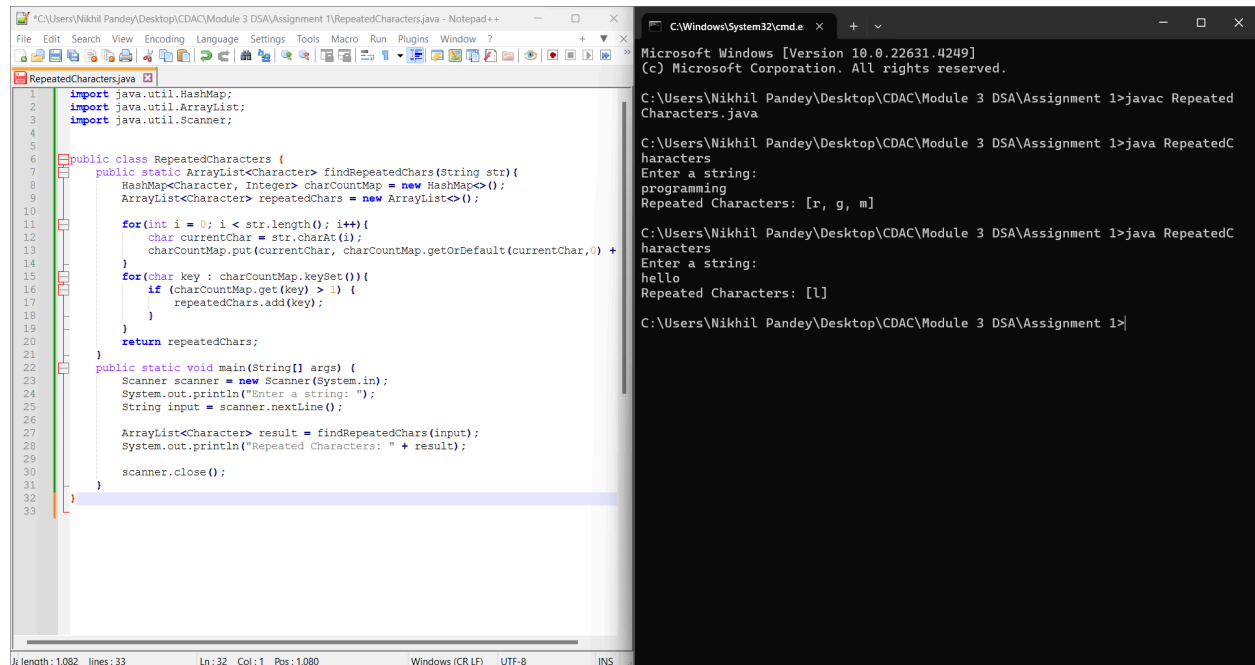
The program uses a HashMap to keep track of the frequency of each character in the input string.

It iterates through the string, updating the count of each character in the map.

After populating the map, the program checks which characters have a count greater than 1 (i.e., repeated characters) and adds them to a list of repeated characters.

The program then prints the repeated characters as output.

-Output



The screenshot displays a Java IDE on the left and a Windows command prompt on the right. The IDE shows the source code for a class named `RepeatedCharacters`. The code imports `HashMap`, `ArrayList`, and `Scanner` from `java.util`. It defines a static method `findRepeatedChars` that takes a string and returns an `ArrayList` of repeated characters. The method uses a `HashMap` to count the frequency of each character and then iterates through the keys to find those with a count greater than 1. The `main` method uses a `Scanner` to take user input and prints the result.

```
1 import java.util.HashMap;
2 import java.util.ArrayList;
3 import java.util.Scanner;
4
5
6 public class RepeatedCharacters {
7     public static ArrayList<Character> findRepeatedChars(String str){
8         HashMap<Character, Integer> charCountMap = new HashMap<>();
9         ArrayList<Character> repeatedChars = new ArrayList<>();
10
11         for(int i = 0; i < str.length(); i++){
12             char currentChar = str.charAt(i);
13             charCountMap.put(currentChar, charCountMap.getOrDefault(currentChar, 0) + 1);
14         }
15         for(char key : charCountMap.keySet()){
16             if (charCountMap.get(key) > 1) {
17                 repeatedChars.add(key);
18             }
19         }
20         return repeatedChars;
21     }
22     public static void main(String[] args) {
23         Scanner scanner = new Scanner(System.in);
24         System.out.println("Enter a string: ");
25         String input = scanner.nextLine();
26
27         ArrayList<Character> result = findRepeatedChars(input);
28         System.out.println("Repeated Characters: " + result);
29
30         scanner.close();
31     }
32 }
33
```

The command prompt shows the compilation and execution of the program. It first runs `javac RepeatedCharacters.java` to compile the code. Then, it runs `java RepeatedCharacters` twice. In the first run, the user enters "programming", and the output is "Repeated Characters: [r, g, m]". In the second run, the user enters "hello", and the output is "Repeated Characters: [l]".

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac Repeated
Characters.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java RepeatedC
haracters
Enter a string:
programming
Repeated Characters: [r, g, m]

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java RepeatedC
haracters
Enter a string:
hello
Repeated Characters: [l]

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity:

The time complexity is **$O(n)$** , where n is the length of the input string. The program iterates through the string once to count the characters and once more to find the repeated characters.

Space Complexity:

The space complexity is **$O(n)$** due to the `HashMap` that stores the frequency of each character and the list that stores the repeated characters.

8. First Non-Repeated Character

Problem: Write a Java program to find the first non-repeated character in a string.

Test Cases:

Input: "stress"

Output: 't'

Input: "aabbcc"

Output: null

Solution:

-Program

```
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Scanner;

public class FirstNonRepeatedCharacter {
    public static Character findFirstNonRepeatedChar(String str){
        LinkedHashMap<Character, Integer> charCountMap = new LinkedHashMap<>();

        for(int i = 0; i < str.length(); i++){
            char currentChar = str.charAt(i);
            charCountMap.put(currentChar, charCountMap.getOrDefault(currentChar, 0) + 1);
        }
        for(Map.Entry<Character, Integer> entry : charCountMap.entrySet()){
            if (entry.getValue() == 1){
                return entry.getKey();
            }
        }
        return null;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string: ");
        String input = scanner.nextLine();
        Character result = findFirstNonRepeatedChar(input);
        if (result != null) {
            System.out.println("The first non-repeated character is : " + result);
        } else {
            System.out.println("No non-repeated character found.");
        }
        scanner.close();
    }
}
```

-Flow Chart

```
/*
    Start
    |
    Enter string str
    |
    Initialize LinkedHashMap to store character counts
    |
    Loop through string and update charCountMap
    |
    Loop through charCountMap to find first character with count = 1
    |
    If found, output first non-repeated character
    |
    If none found, output "No non-repeated character found"
    |
    End
*/
```

-Explanation

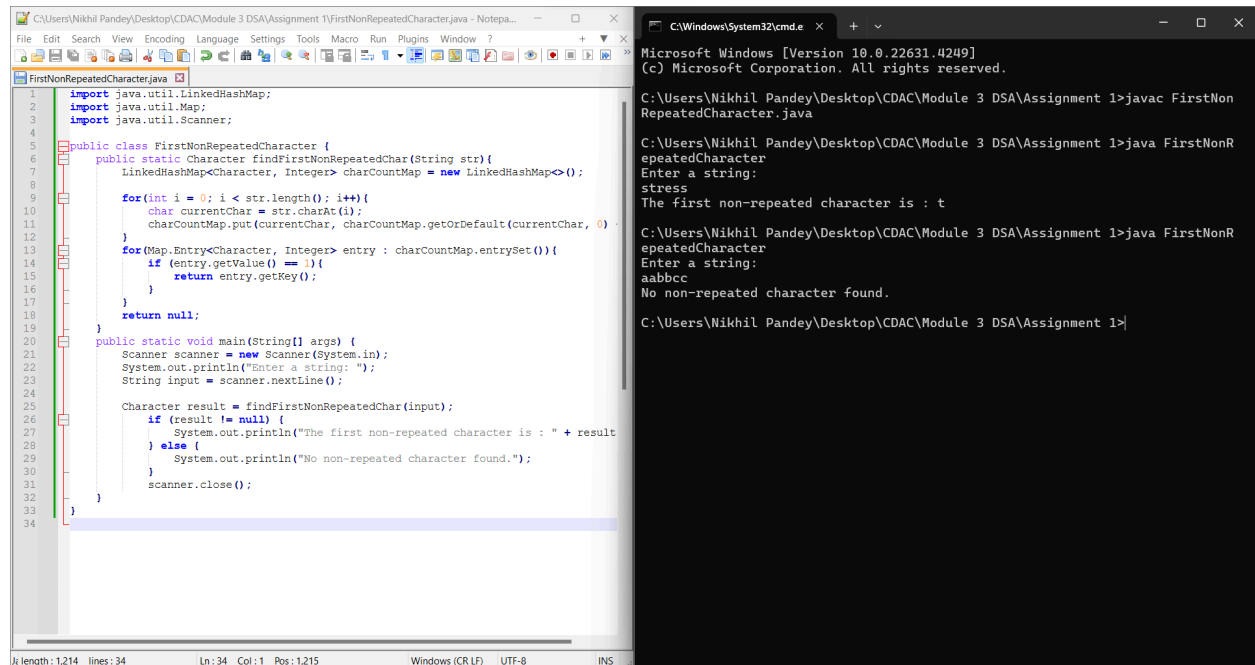
The program uses a LinkedHashMap to store the count of each character while maintaining the order of insertion.

The string is first looped through to update the count of each character in the map.

The program then loops through the map entries to find the first character that has a count of 1 (i.e., a non-repeated character).

If no such character is found, the program returns null and prints a message indicating that no non-repeated character exists.

-Output



The screenshot displays a Java IDE on the left and a Windows command prompt on the right. The IDE shows the source code for a class named `FirstNonRepeatedCharacter`. The code imports `java.util.LinkedHashMap`, `java.util.Map`, and `java.util.Scanner`. It defines a static method `findFirstNonRepeatedChar` that takes a string and returns the first non-repeated character. The method uses a `LinkedHashMap` to count the occurrences of each character. The `main` method uses a `Scanner` to take user input and prints the result.

```
1 import java.util.LinkedHashMap;
2 import java.util.Map;
3 import java.util.Scanner;
4
5 public class FirstNonRepeatedCharacter {
6     public static Character findFirstNonRepeatedChar(String str){
7         LinkedHashMap<Character, Integer> charCountMap = new LinkedHashMap<>();
8
9         for(int i = 0; i < str.length(); i++){
10             char currentChar = str.charAt(i);
11             charCountMap.put(currentChar, charCountMap.getOrDefault(currentChar, 0) + 1);
12         }
13         for(Map.Entry<Character, Integer> entry : charCountMap.entrySet()){
14             if (entry.getValue() == 1){
15                 return entry.getKey();
16             }
17         }
18         return null;
19     }
20
21     public static void main(String[] args) {
22         Scanner scanner = new Scanner(System.in);
23         System.out.println("Enter a string: ");
24         String input = scanner.nextLine();
25
26         Character result = findFirstNonRepeatedChar(input);
27         if (result != null) {
28             System.out.println("The first non-repeated character is : " + result);
29         } else {
30             System.out.println("No non-repeated character found.");
31         }
32         scanner.close();
33     }
34 }
```

The command prompt shows the compilation and execution of the program. It first compiles the `FirstNonRepeatedCharacter.java` file. Then, it runs the program, which prompts the user to enter a string. The user enters `stress`, and the program outputs `The first non-repeated character is : t`. In a subsequent run, the user enters `aabbcc`, and the program outputs `No non-repeated character found.`

-Time and Space complexity

Time Complexity:

The time complexity is **$O(n)$** , where n is the length of the input string. The program loops through the string twice: once to count the characters and once to find the first non-repeated character.

Space Complexity:

The space complexity is **$O(n)$** due to the `LinkedHashMap` used to store the character counts.

9. Integer Palindrome

Problem: Write a Java program to check if a given integer is a palindrome.

Test Cases:

Input: 121

Output: true

Input: -121

Output: false

Solution:

-Program

```
import java.util.Scanner;

public class IntegerPalindrome {
    public static boolean isPalindrome(int num){
        if (num < 0) {
            return false;
        }
        int originalNum = num;
        int reverseNum = 0;

        while (num != 0) {
            int lastDigit = num % 10;
            reverseNum = reverseNum * 10 + lastDigit;
            num = num / 10;
        }
        return originalNum == reverseNum;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int num = scanner.nextInt();

        boolean result = isPalindrome(num);
        if (result) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }
        scanner.close();
    }
}
```


-Flow Chart

```
/*
    Start
    |
    Enter integer num
    |
    Is num < 0? ---> Yes --> Output: Not a palindrome
    |
    No
    |
    Initialize reversedNum = 0, originalNum = num
    |
    While num != 0
    |
    Get lastDigit = num % 10
    |
    Update reversedNum = reversedNum * 10 + lastDigit
    |
    Update num = num / 10
    |
    Check if originalNum == reversedNum
    |
    If true --> Output: Palindrome
    If false --> Output: Not a palindrome
    |
    End
*/
```

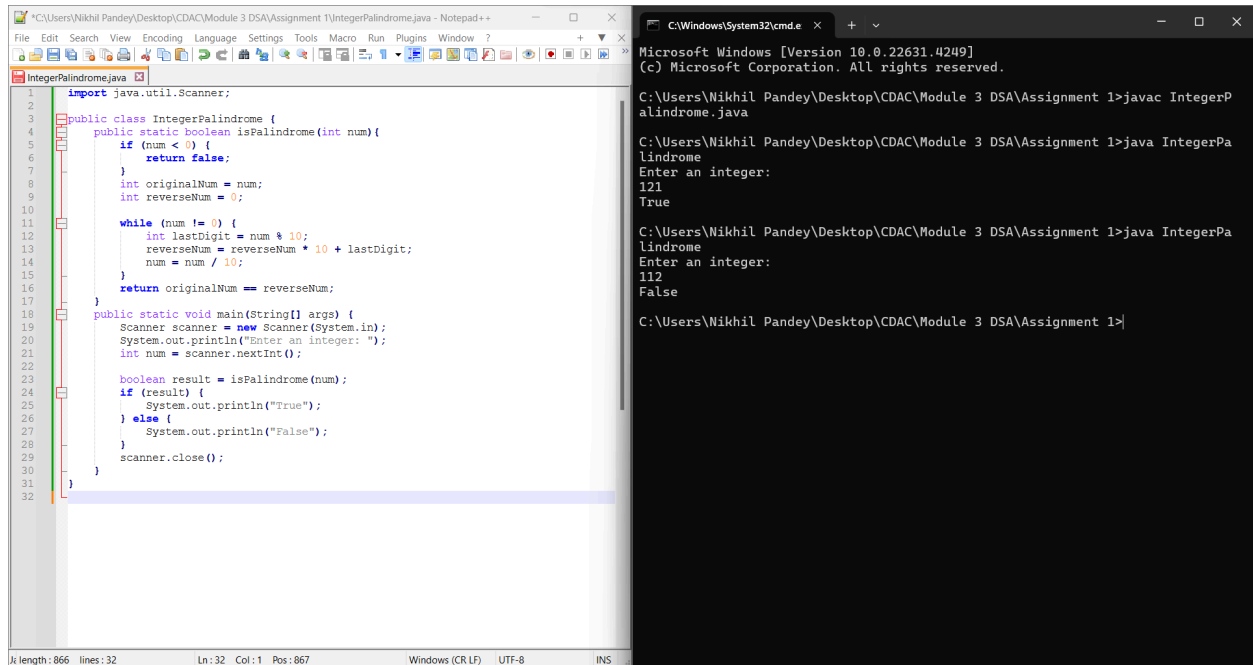
-Explanation

Negative numbers are immediately ruled out as non-palindromes because a palindrome must read the same forward and backward, and negative signs make this impossible.

The program reverses the digits of the given integer. It extracts each digit using the modulo operator (%), constructs the reversed number, and compares it with the original integer.

If the original number equals its reversed version, it is considered a palindrome; otherwise, it is not.

-Output



The image shows a Java IDE window on the left and a Windows command prompt on the right. The IDE window displays the source code for a Java class named `IntegerPalindrome`. The code imports `java.util.Scanner` and defines a `main` method that takes an integer input and checks if it is a palindrome. The logic involves reversing the digits of the number and comparing it to the original number. The command prompt shows the execution of the program, with the user entering the number 121, which is correctly identified as a palindrome (True), and the number 112, which is correctly identified as not a palindrome (False).

```
1 import java.util.Scanner;
2
3 public class IntegerPalindrome {
4     public static boolean isPalindrome(int num){
5         if (num < 0) {
6             return false;
7         }
8         int originalNum = num;
9         int reverseNum = 0;
10
11         while (num != 0) {
12             int lastDigit = num % 10;
13             reverseNum = reverseNum * 10 + lastDigit;
14             num = num / 10;
15         }
16         return originalNum == reverseNum;
17     }
18     public static void main(String[] args) {
19         Scanner scanner = new Scanner(System.in);
20         System.out.println("Enter an integer: ");
21         int num = scanner.nextInt();
22
23         boolean result = isPalindrome(num);
24         if (result) {
25             System.out.println("True");
26         } else {
27             System.out.println("False");
28         }
29         scanner.close();
30     }
31 }
32
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac IntegerP
alindrome.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java IntegerPa
lindrome
Enter an integer:
121
True

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java IntegerPa
lindrome
Enter an integer:
112
False

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity:

The time complexity is **$O(d)$** , where d is the number of digits in the integer. The program iterates over each digit to reverse the number.

Space Complexity:

The space complexity is **$O(1)$** , since the program only uses a few integer variables (`num`, `reversedNum`, `lastDigit`, and `originalNum`) regardless of the size of the input number.

10. Leap Year

Problem: Write a Java program to check if a given year is a leap year.

Test Cases:

Input: 2020

Output: true

Input: 1900

Output: false

Solution:

-Program

```
import java.util.Scanner;

public class LeapYear {

    public static boolean isLeapYear(int year) {

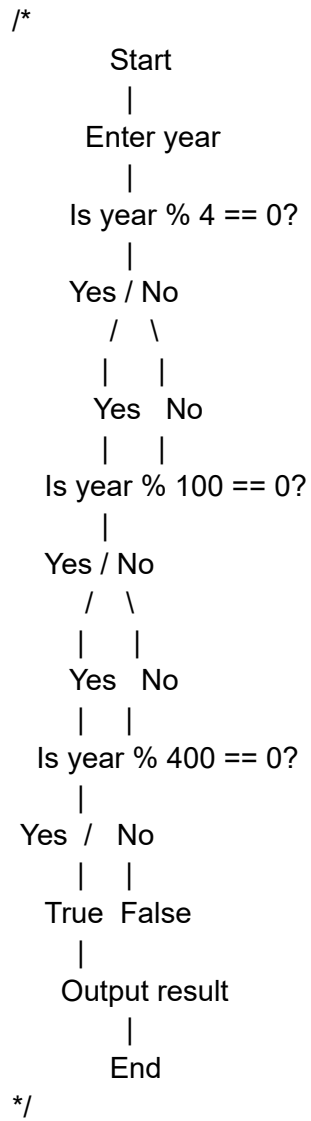
        if (year % 4 == 0) {
            if (year % 100 == 0) {
                return year % 400 == 0;
            }
            return true;
        }
        return false;
    }

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

        boolean result = isLeapYear(year);
        if (result) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }

        scanner.close();
    }
}
```

-Flow Chart



-Explanation

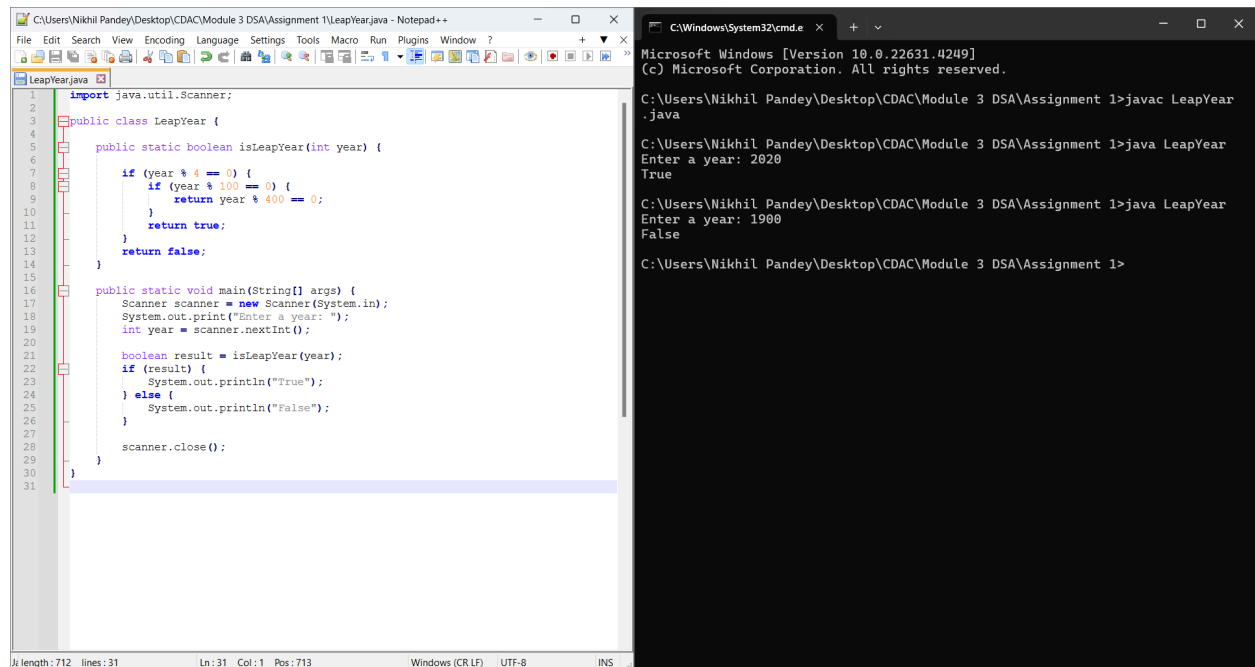
A year is determined to be a leap year based on the following rules:

It is divisible by 4.

If it is divisible by 100, it must also be divisible by 400 to be considered a leap year.

The program checks these conditions using simple modulo operations and returns true or false based on the results.

-Output



```
1 import java.util.Scanner;
2
3 public class LeapYear {
4
5     public static boolean isLeapYear(int year) {
6
7         if (year % 4 == 0) {
8             if (year % 100 == 0) {
9                 return year % 400 == 0;
10            }
11            return true;
12        }
13        return false;
14    }
15
16    public static void main(String[] args) {
17        Scanner scanner = new Scanner(System.in);
18        System.out.print("Enter a year: ");
19        int year = scanner.nextInt();
20
21        boolean result = isLeapYear(year);
22        if (result) {
23            System.out.println("True");
24        } else {
25            System.out.println("False");
26        }
27
28        scanner.close();
29    }
30
31 }
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>javac LeapYear
.java
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java LeapYear
Enter a year: 2020
True
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>java LeapYear
Enter a year: 1900
False
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 1>
```

-Time and Space complexity

Time Complexity:

The time complexity is **O(1)** since the program consists of a constant number of operations, regardless of the input year.

Space Complexity:

The space complexity is **O(1)** as well, as only a few variables are used to hold the input year and the result.