C-DAC Mumbai

Date 25/09/2024

Subject: Algorithm and Data Structure
Assignment 1

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

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

1. Armstrong Number

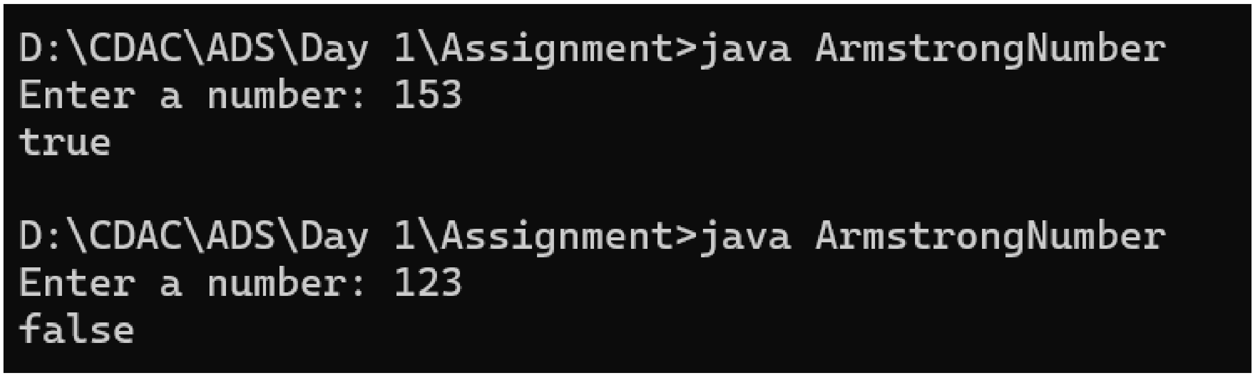
Problem: Write a Java program to check if a given number is an Armstrong number.

Test Cases:

Input: 153
Output: true
Input: 123
Output: false

Program:

```
import java.util.Scanner;

public class ArmstrongNumber {
    public static boolean isArmstrong(int num) {
        int original = num, result = 0;
        int n = String.valueOf(num).length();

        while (num != 0) {
            int digit = num % 10;
            result += Math.pow(digit, n);
            num /= 10;
        }

        return result == original;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        System.out.println(isArmstrong(num));
    }
}
/*
Explanation:

1. Input: A number num
2. Find the number of digits n
3. For each digit, raise it to the power n and sum the values
4. If the sum equals the original number, it is an Armstrong number.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java ArmstrongNumber
Enter a number: 153
true

D:\CDAC\ADS\Day 1\Assignment>java ArmstrongNumber
Enter a number: 123
false
```

Space complexity: O (1)

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

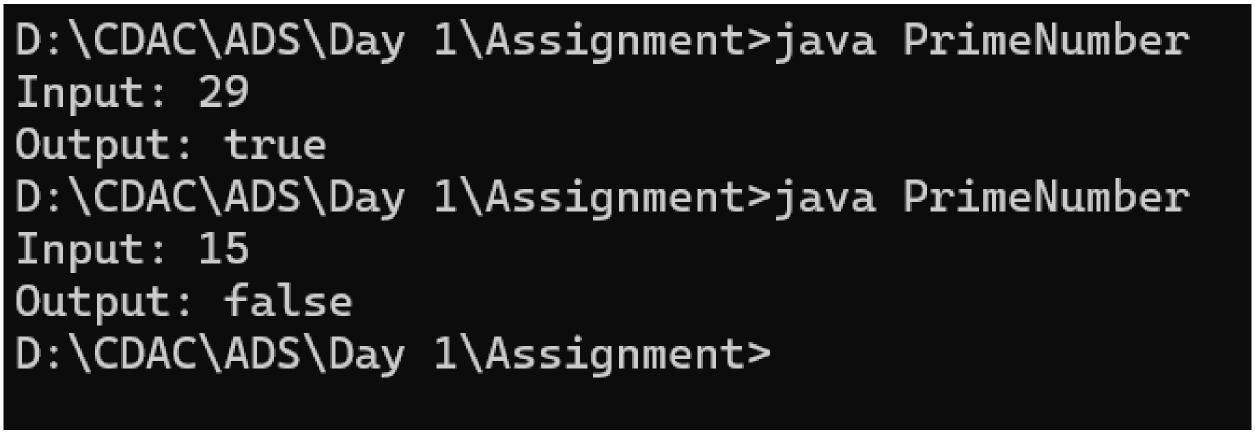
Test Cases:

Input: 29
Output: true
Input: 15
Output: false

Program:

```
import java.util.*;
class PrimeNumber
{
      public static void main(String[] args)
      {
            Scanner sc = new Scanner(System.in);
```

```
        boolean isprime= true;
        System.out.print("Input: ");
        int num =sc.nextInt();
        System.out.print("Output: ");
        if(num < 2)
        {
            isprime = false;
        }
         else
        {
            for(int i=2; i <= num/2; i++)
                {
                if(num % i == 0)
                {
                 isprime = false;
                 break;
                }
            }
        }
        System.out.print(isprime);
    }
}
/*
Explanation:
1. If the number is lesser than 2.
2. If the number has any factors other than 1 and itself.
3. If either of the above two conditions are satisfied, the number is not
a Prime.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java PrimeNumber
Input: 29
Output: true
D:\CDAC\ADS\Day 1\Assignment>java PrimeNumber
Input: 15
Output: false
D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(n)
Space Complexity: O (1)

3. Factorial

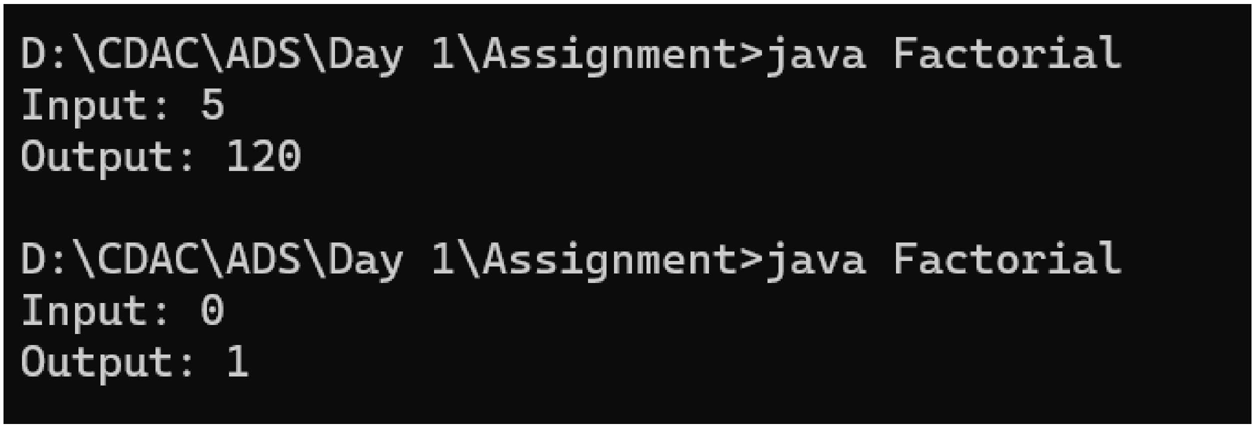
Problem: Write a Java program to compute the factorial of a given number.

Test Cases:

Input: 5
Output: 120
Input: 0
Output: 1

Program:

```
import java.util.*;
class Factorial{
      static int fact(int num){
            if(num<=1)
                  return 1;
            return num*fact(num-1);
      }
      public static void main(String[] args){
            Scanner sc = new Scanner(System.in);
            System.out.print("Input: ");
            int num = sc.nextInt();
            int fact = fact(num);
            System.out.println("Output: "+fact);

      }
}
/*Explanation:
1. Call function fact(num)
2. Set base case when num<=1 return 1
3. Other cases return num * fact(num-1);
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java Factorial
Input: 5
Output: 120

D:\CDAC\ADS\Day 1\Assignment>java Factorial
Input: 0
Output: 1
```

Time Complexity: O(n)
Space Complexity: O(n) due to the recursive calls, where n is the input number.

4. Fibonacci Series

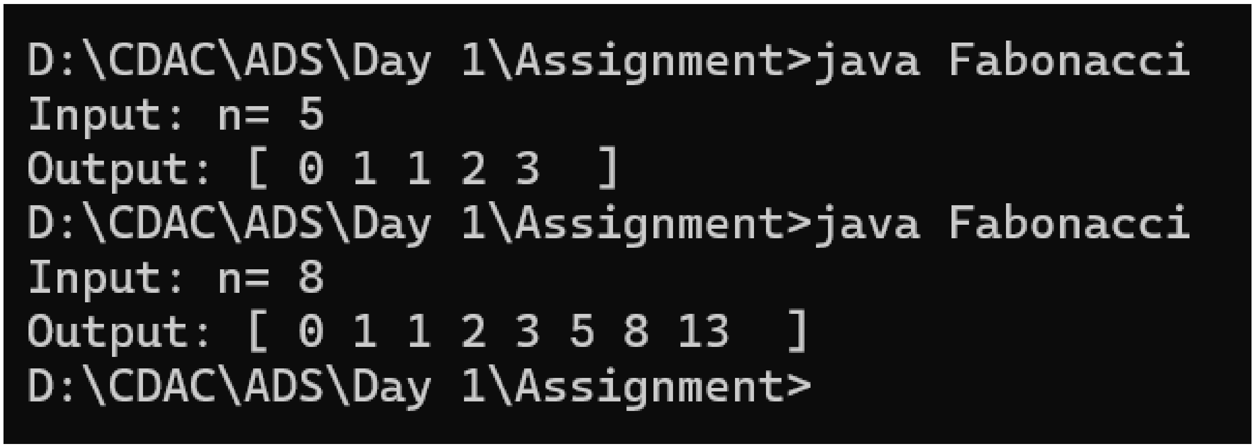
Problem: Write a Java program to print the first n numbers in the Fibonacci series.

Test Cases:

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

Program:

```
import java.util.*;
class Fabonacci{
      static int fab(int n){
            if(n<=1)  //base condition
            {
                  return n;
            }
            return fab(n-1)+fab(n-2);
      }
      public static void main(String args[])
      {
            Scanner sc = new Scanner(System.in);
            System.out.print("Input: n= ");
            int num = sc.nextInt();
            System.out.print("Output: [ ");
            for(int i=0;i<num;i++)
            {
                  System.out.print(fab(i)+" ");
            }
            System.out.print(" ]");
      }
}

/*
Explanation:
1. The fab() function calculates Fibonacci numbers using recursion.
2. The program asks the user to enter a number (num).
3. It then prints the first num Fibonacci numbers using a loop.
4. The output is displayed in square brackets.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java Fabonacci
Input: n= 5
Output: [ 0 1 1 2 3  ]
D:\CDAC\ADS\Day 1\Assignment>java Fabonacci
Input: n= 8
Output: [ 0 1 1 2 3 5 8 13  ]
D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(2^n)
Space Complexity: O(n)

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

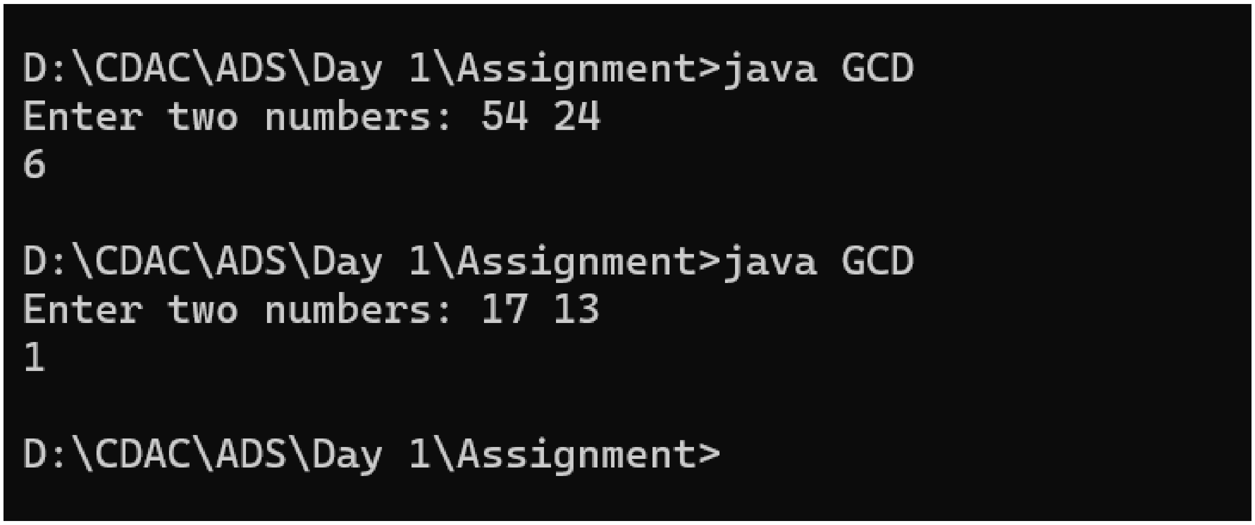
Test Cases:

Input: a = 54, b = 24
Output: 6
Input: a = 17, b = 13
Output: 1

Program:

```java
import java.util.*;

public class GCD {
    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
        int a = sc.nextInt();
        int b = sc.nextInt();
        System.out.println(gcd(a, b));
    }
}
```

```
/*Explanation:
1. Take two numbers a and b
2. Use the formula where GCD(a, b) = GCD(b, a % b) until b becomes 0.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java GCD
Enter two numbers: 54 24
6

D:\CDAC\ADS\Day 1\Assignment>java GCD
Enter two numbers: 17 13
1

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(log(min(a,b)))
Space Complexity: O(log(min(a,b)))   due to the recursion depth.

6. Find Square Root

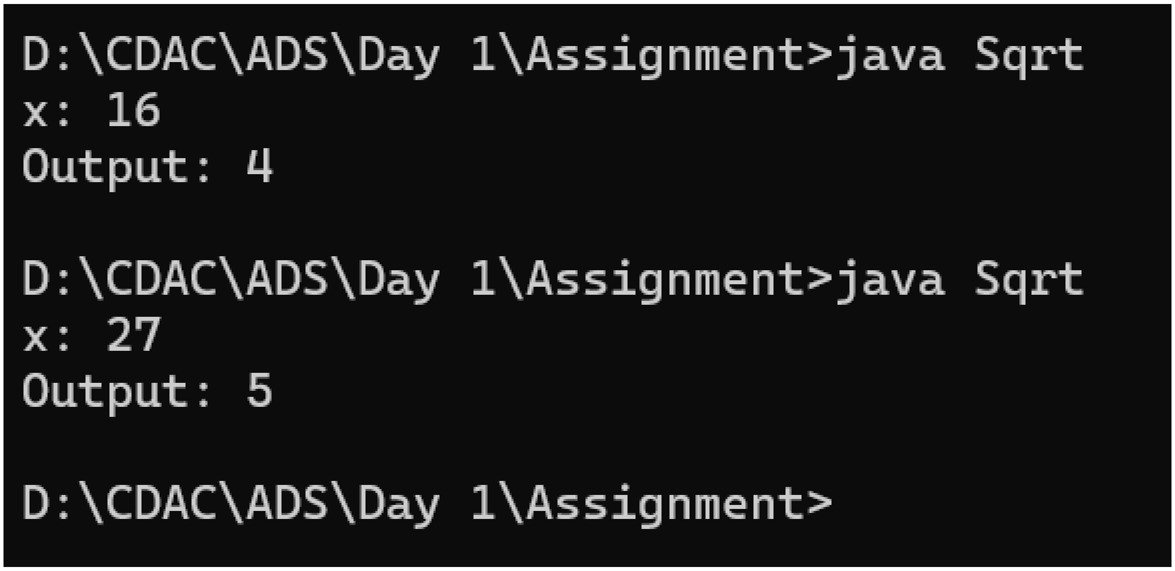
Problem: Write a Java program to find the square root of a given number (using integer approximation).

Test Cases:

Input: x = 16
Output: 4
Input: x = 27
Output: 5

Program:

```
import java.util.*;
class Sqrt {
    static int sqrt(int x)
    {
        if (x == 0 || x == 1)
            return x;
        int i = 1, result = 1;

        while (result <= x) {
            i++;
            result = i * i;
        }
        return i - 1;
    }

    public static void main(String[] args)
    {
           Scanner sc = new Scanner(System.in);
           System.out.print("x: ");
        int x = sc.nextInt();
        System.out.println("Output: "+sqrt(x));
    }
}
/*
Explanation:
1. Create a variable i and write base cases, (i.e when the given number is
0 or 1).
2. Run a loop until i*i <= n, where n is the given number. Increment i by
1.
3. The square root of the number is i â€" 1
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java Sqrt
x: 16
Output: 4

D:\CDAC\ADS\Day 1\Assignment>java Sqrt
x: 27
Output: 5

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(√x)
Space Complexity: O(1)

7. Find Repeated Characters in a String

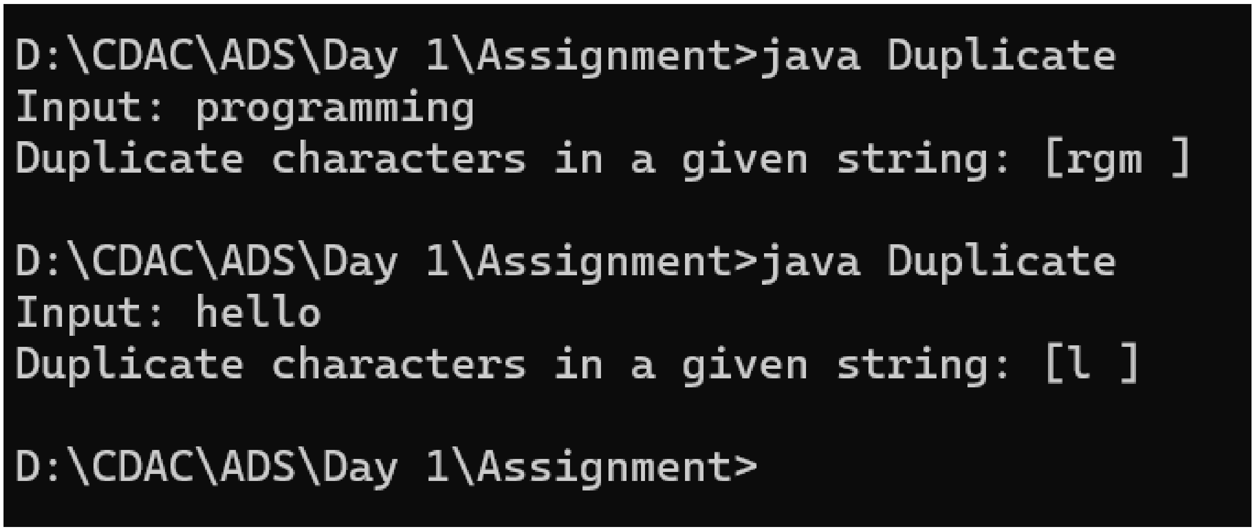
Problem: Write a Java program to find all repeated characters in a string.

Test Cases:

Input: "programming"
Output: ['r', 'g', 'm']
Input: "hello"
Output: ['l']

Program:

```
import java.util.*;
class Duplicate {
     public static void main(String[] args) {
           Scanner sc = new Scanner(System.in);
           System.out.print("Input: ");
        String string1 = sc.nextLine();
        int count;

        char string[] = string1.toCharArray();

        System.out.print("Duplicate characters in a given string: [");

        for(int i = 0; i <string.length; i++) {
            count = 1;
            for(int j = i+1; j <string.length; j++) {
                if(string[i] == string[j] && string[i] != ' ') {
                    count++;

                    string[j] = '0';
                }
            }

            if(count > 1 && string[i] != '0')
                System.out.print(string[i]);
        }
           System.out.println(" ]");
    }
}
/*Explanation:
1. Converts given string into character array
2. Counts each character present in the string
3. Set string[j] to 0 to avoid printing visited character
4. A character is considered as duplicate if count is greater than 1
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java Duplicate
Input: programming
Duplicate characters in a given string: [rgm ]

D:\CDAC\ADS\Day 1\Assignment>java Duplicate
Input: hello
Duplicate characters in a given string: [l ]

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(n^2)
Space Complexity: O(n)

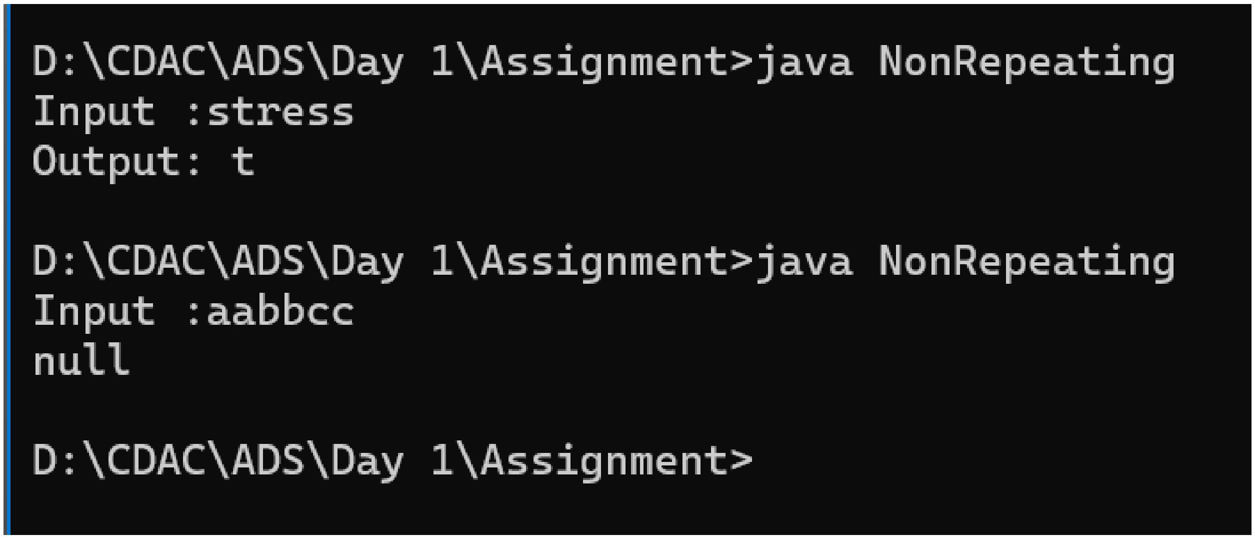
8. First Non-Repeated Character
Problem: Write a Java program to find the first non-repeated character in a string.

Test Cases:

Input: "stress"
Output: 't'
Input: "aabbcc"
Output: null

Program:

```
import java.util.*;
class NonRepeating
{
    public static void main(String args[])
    {
        Scanner s=new Scanner(System.in);
        System.out.print("Input :");
        String word = s.next();
        boolean flag = true;

        for(char i :word.toCharArray())
        {
            if (word.indexOf(i) == word.lastIndexOf(i))
            {
                System.out.println("Output: "+ i);
                flag = false;
                break;
            }
        }
        if(flag== true){
            System.out.println("null");
            }
    }
}
/*
Explanation:

1. Input: The program reads a word from the user.
2. It checks each character and compares its first and last occurrence. If
they are the same, it's the first non-repeated character.
3. Output: It prints the first non-repeated character, or "null" if none
is found.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java NonRepeating
Input :stress
Output: t

D:\CDAC\ADS\Day 1\Assignment>java NonRepeating
Input :aabbcc
null

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(n^2) n=length of string
Space Complexity:O(1)

9. Integer Palindrome

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

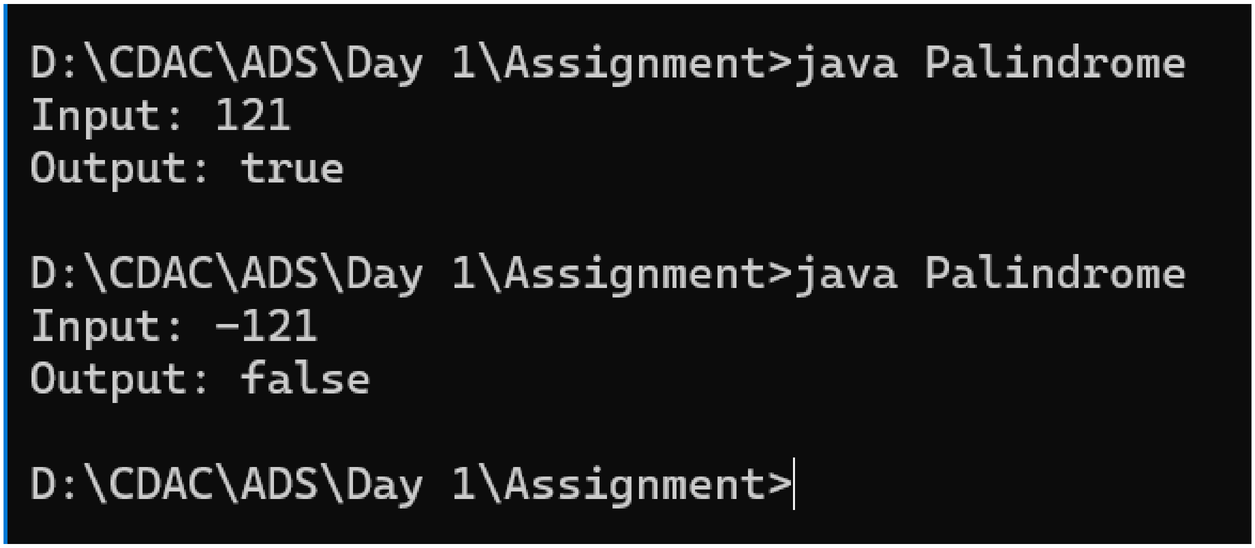
Test Cases:

Input: 121
Output: true
Input: -121

Output: false

Program:

```
import java.util.Scanner;

class Palindrome {
    public static boolean isPalindrome(int num) {
        if (num < 0) return false;
        int reversed = 0, ori = num;

        while (num != 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }

        return ori == reversed;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input: ");
        int num = sc.nextInt();
        System.out.println("Output: "+isPalindrome(num));
    }
}
/*
Explanation:
1. Input: A number num
2. Reverse the digits of the number and compare with the original.
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java Palindrome
Input: 121
Output: true

D:\CDAC\ADS\Day 1\Assignment>java Palindrome
Input: -121
Output: false

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: $O(\log_{10}(n))$
Space Complexity: $O(1)$

10. Leap Year

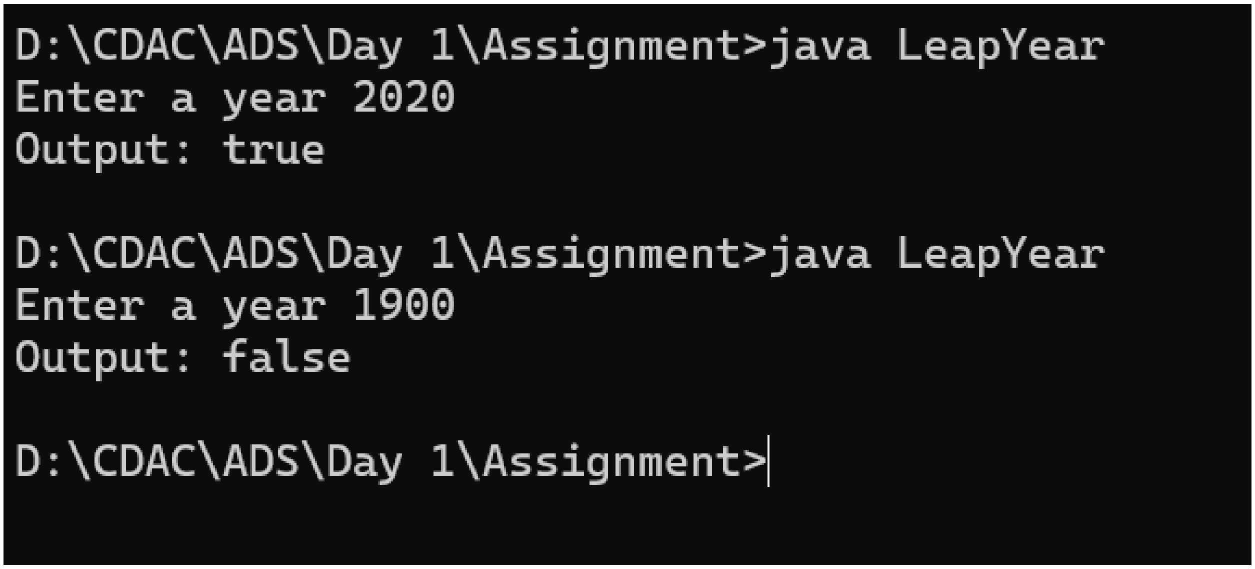
Problem: Write a Java program to check if a given year is a leap year.

Test Cases:

Input: 2020
Output: true
Input: 1900
Output: false

Program:

```
import java.util.*;
class LeapYear{
    static boolean check(int year){
        if(year % 400 == 0)
        {
            return true;
        }
        else if(year % 4 == 0 && year % 100 != 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a year");
        int year = sc.nextInt();
        System.out.println("Output: " + check(year));

    }
}
/*
Explanation:
1. take year as input.
2. Check if the year is divisible by 400 = leap year
3. Check if the year is divisible by 4 but not by 100 = leap year
4. Otherwise, it's not a leap year
*/
```



```
D:\CDAC\ADS\Day 1\Assignment>java LeapYear
Enter a year 2020
Output: true

D:\CDAC\ADS\Day 1\Assignment>java LeapYear
Enter a year 1900
Output: false

D:\CDAC\ADS\Day 1\Assignment>
```

Time Complexity: O(1)
Space Complexity:O(1)