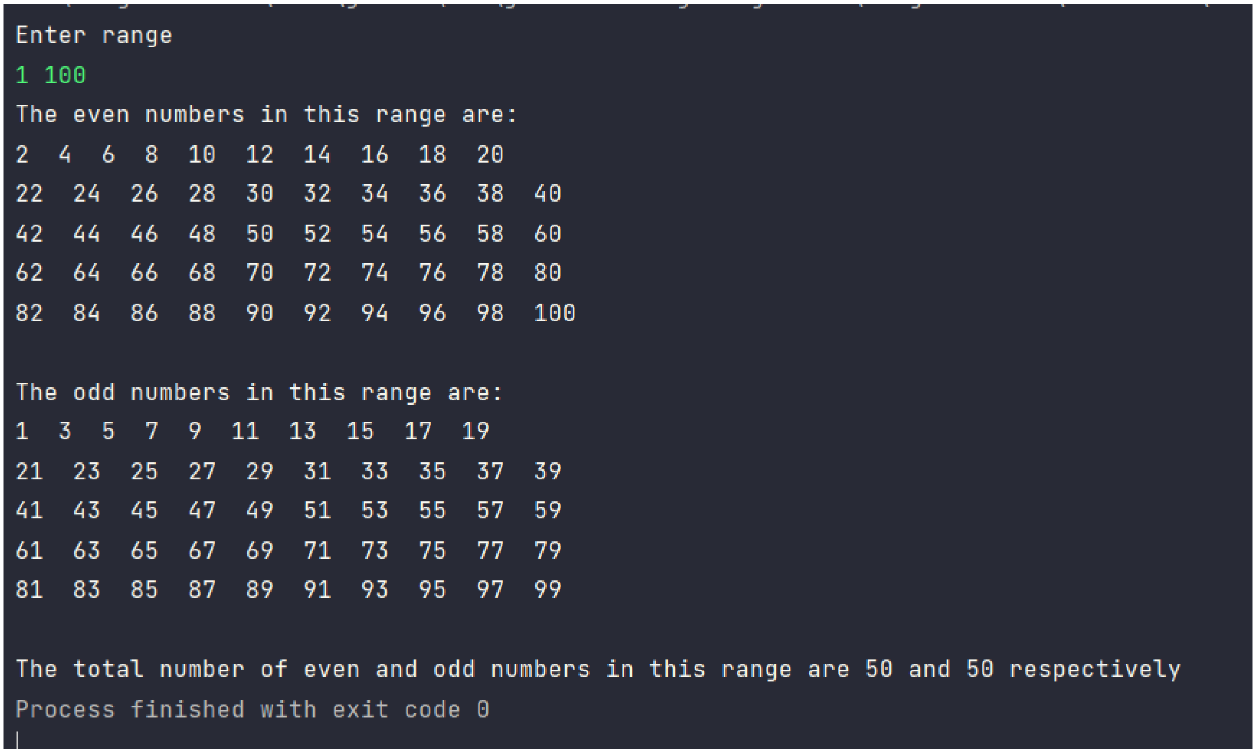
| | |
|---|---|
| **Name** | Shubhan Singh |
| **UID no.** | 2022300118 |
| **Experiment No.** | 1 |

| **Program 1** | |
|---|---|
| **PROBLEM STATEMENT :** | **Write a program to print all even and odd numbers in the range of 1-100 as well as the total number of even and odd numbers in the given range.**<br>**As an extra, the range may be inputted from the user.** |
| **THEORY:** | An even number is a number which is divisible by 2, while odd numbers are not divisible by 2.<br>Taking user input:<br>In Java, the Scanner class is used to take input from the user via the console. To use the Scanner class, you first need to create an instance of the class. This is done by declaring a new Scanner object and passing in the System.in parameter, which tells Java to use the standard input stream. Once you have created an instance of the Scanner class, you can use its various methods to read input from the user. For example, the nextLine() method is used to read a line of text, while the nextInt() method is used to read an integer value.<br>Writing output to console:<br>In Java, printing output to the console is done using the System.out.println() method. This method takes a string or any other data type as an argument and outputs it to the console followed by a newline character. If you want to print a message without adding a newline character, you can use the System.out.print() method instead. You can also use the format() method to format your output in a specific way, similar to printf() in C, or use printf() itself. Additionally, you can redirect output to a file or another output stream using the System.setOut() method |

**PROGRAM:**

```
import java.util.Scanner;//Importing util library for scanner
class

public class Evenodd {
    public static void main(String[] args) {
        int start,end,evencount=0,oddcount=0;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter range");
        start=sc.nextInt();
        end=sc.nextInt();//Taking range from user input
        System.out.println("The even numbers in this range
are:");
        for(int i=start;i<=end;i++){
            if(i%2==0){//checking if a number is even
                evencount++;//keeping count of even numbers
                System.out.printf("%d  ",i);
                if(evencount%10==0){
                    System.out.print("\n");//printing a new line
after every 10 numbers
                }
            }
        }
        System.out.println("\nThe odd numbers in this range
are:");
        for(int i=start;i<=end;i++){//checking if a number is
odd
            if(i%2!=0){
                oddcount++;//keeping count of odd numbers
                System.out.printf("%d  ",i);
                if(oddcount%10==0){
                    System.out.print("\n");//printing a new line
after every 10 numbers
                }
            }
        }
        System.out.printf("\nThe total number of even and odd
numbers in this range are %d and %d
respectively",evencount,oddcount);
    }
}
```

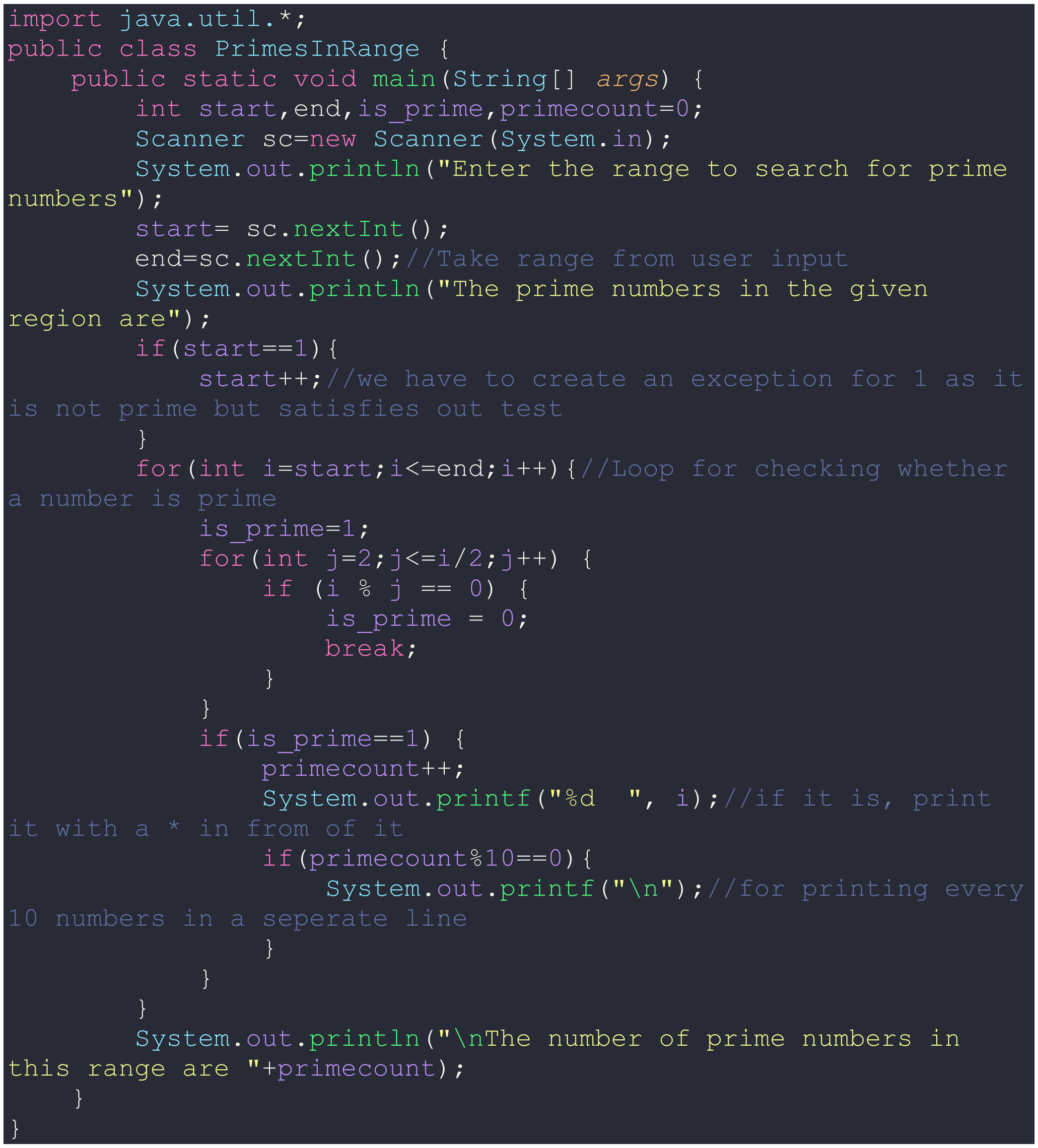
**RESULT:**



```
Enter range
1 100
The even numbers in this range are:
2  4  6  8  10  12  14  16  18  20
22  24  26  28  30  32  34  36  38  40
42  44  46  48  50  52  54  56  58  60
62  64  66  68  70  72  74  76  78  80
82  84  86  88  90  92  94  96  98  100

The odd numbers in this range are:
1  3  5  7  9  11  13  15  17  19
21  23  25  27  29  31  33  35  37  39
41  43  45  47  49  51  53  55  57  59
61  63  65  67  69  71  73  75  77  79
81  83  85  87  89  91  93  95  97  99

The total number of even and odd numbers in this range are 50 and 50 respectively
Process finished with exit code 0
```

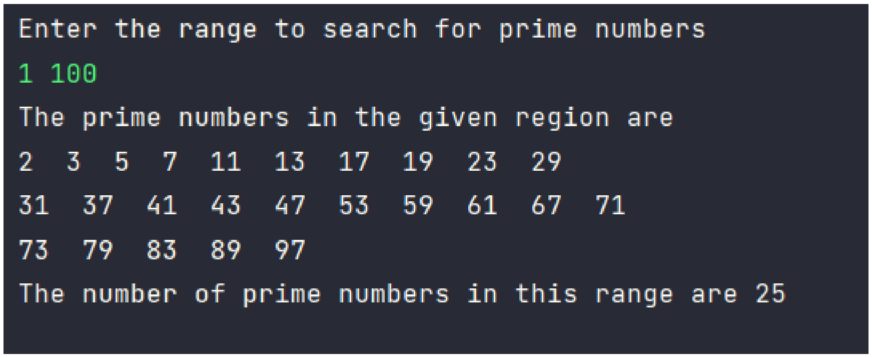
| **Program 2** | |
|---|---|
| **PROBLEM STATEMENT :** | **To check for all prime numbers in a given range.** |
| **THEORY:** | In this Program, we have used a for loop to cycle through all the values within the range and apply the same set of operations to them to check whether they are prime or not.<br><br>For loops in java:<br>In Java, a for loop is a control structure that allows you to iterate over a block of code a specified number of times. The for loop consists of three parts: the initialization, the condition, and the increment. The initialization sets the initial value of the loop variable, the condition is a Boolean expression that is evaluated before each iteration, and the increment updates the loop variable at the end of each iteration. Inside the loop, you can execute any code that you want to repeat. You can also use the break and continue statements to control the flow of the loop. For loops are particularly useful when you need to perform a certain operation a specific number of times. It is important to ensure that the loop condition is properly set to avoid infinite loops that could cause your program to crash. |

**PROGRAM:**



```
import java.util.*;
public class PrimesInRange {
    public static void main(String[] args) {
        int start,end,is_prime,primecount=0;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the range to search for prime
numbers");
        start= sc.nextInt();
        end=sc.nextInt();//Take range from user input
        System.out.println("The prime numbers in the given
region are");
        if(start==1){
            start++;//we have to create an exception for 1 as it
is not prime but satisfies out test
        }
        for(int i=start;i<=end;i++){//Loop for checking whether
a number is prime
            is_prime=1;
            for(int j=2;j<=i/2;j++) {
                if (i % j == 0) {
                    is_prime = 0;
                    break;
                }
            }
            if(is_prime==1) {
                primecount++;
                System.out.printf("%d  ", i);//if it is, print
it with a * in from of it
                if(primecount%10==0){
                    System.out.printf("\n");//for printing every
10 numbers in a seperate line
                }
            }
        }
        System.out.println("\nThe number of prime numbers in
this range are "+primecount);
    }
}
```

**RESULT:**



```
Enter the range to search for prime numbers
1 100
The prime numbers in the given region are
2  3  5  7  11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97
The number of prime numbers in this range are 25
```

## Program 3

**PROBLEM** **Write a program to print all Armstrong numbers and Prime Numbers in**

**STATEMENT:**

**the range inputted by the user. Also print the total count of Prime Numbers in the given range.**

**Use the concept of classes and objects.**

**THEORY:**

An Armstrong number is a number in any given number base, which forms the total of the same number, when each of its digits is raised to the power of the number of digits in the number.
For example, using a simple number 153 and the decimal system, we see there are 3 digits in it. If we do a simple mathematical operation of raising each of its digits to the power of 3, and then totalling the sum obtained, we get 153. That is 1 to the power of 3, 5 to the power of 3, 3 to the power of three is 1+125+27= 153.

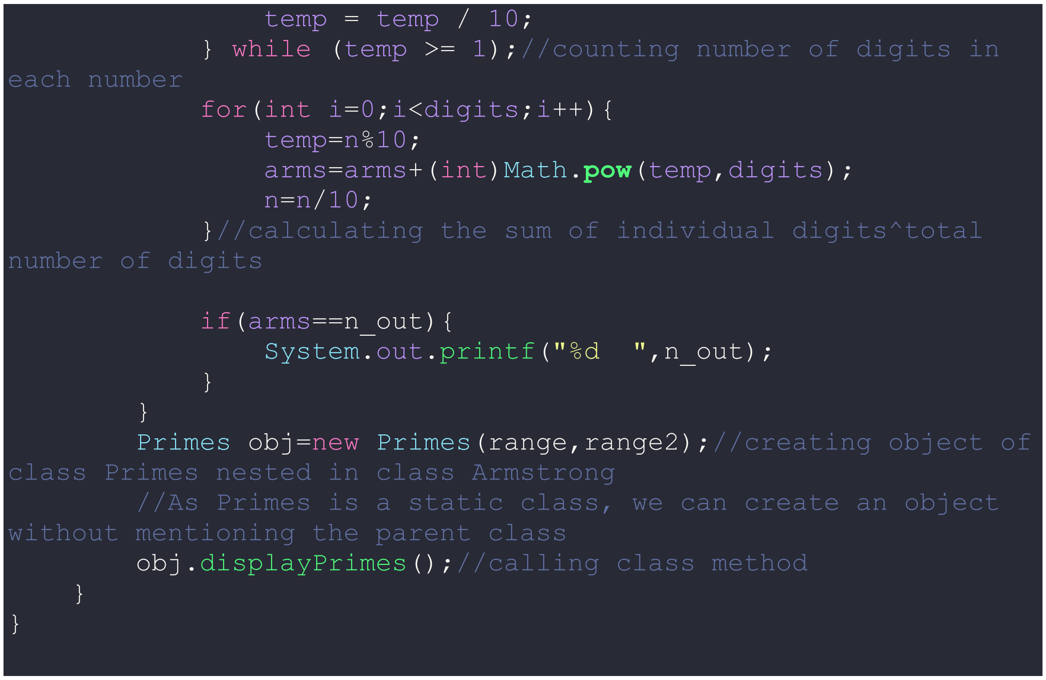
Here we use the Math.pow() method from the Math class.
In Java, the Math.pow() method is a built-in function that returns the result of raising a base to a specified power. The method takes two arguments: the base and the exponent. The base and the exponent can be any numeric data type, including double, float, long, and int. The Math.pow() method returns a double value that represents the result of raising the base to the specified power. For example, Math.pow(2, 3) would return 8, as 2 raised to the power of 3 equals 8.

Here we used another class to print the prime numbers. It is a static class, a static class is one which does not need a reference to its outer class to be instantiated. It can only access the static members of the outer class and it is easier to initialize objects with static classes.
We used a parameterized constructor to pass the range to the Primes class, as it cannot access the members of the main class, since we have not established any inheritance there.
We use the displayPrimes method inside the Primes class to display the prime numbers and their count in the given range, it is a void function and has no parameters as it can already access the needed variables in the class.
The new keyword is used to create an object of a class.

**PROGRAM:**

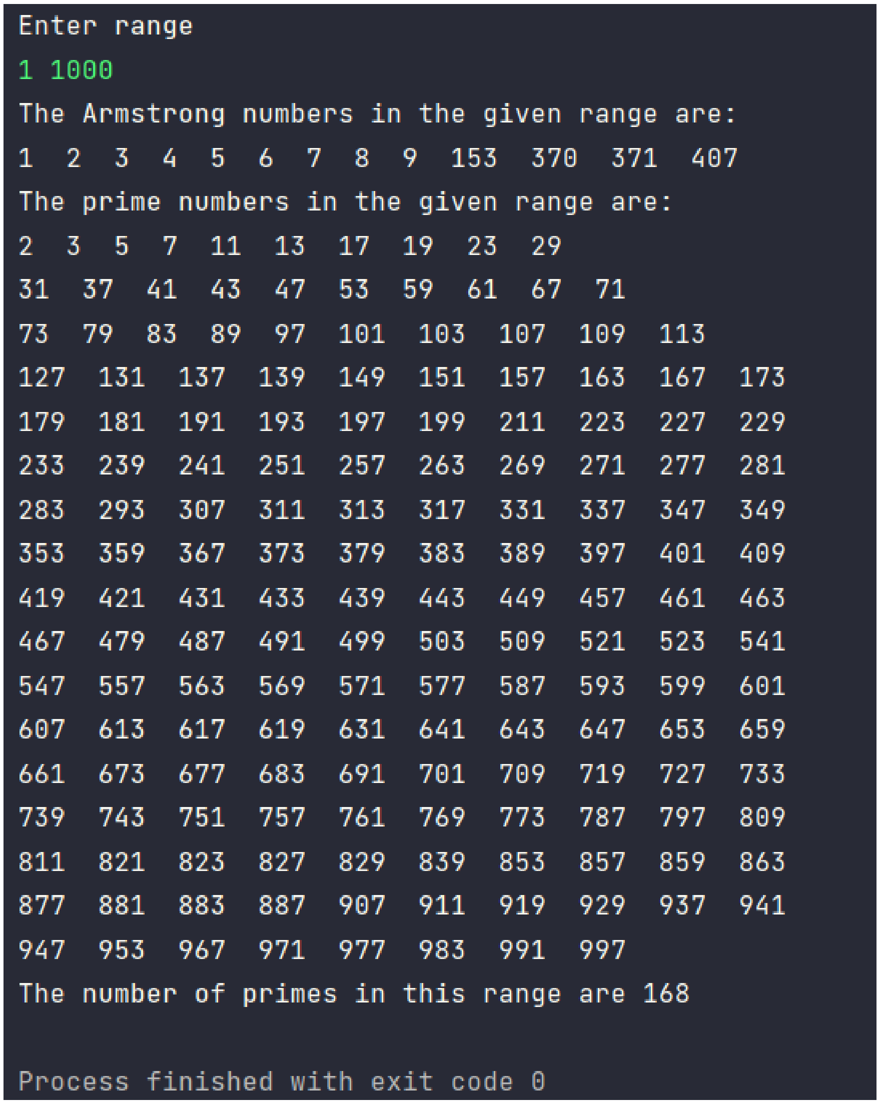
```
import java.util.*;      //Importing util and math class for scanner class and pow function
import java.lang.Math;
public class Armstrong{
    public static class Primes{//We declared it as static class as we do not have multiple instances of the parent class and thus it would be easier to create objects this way.
```

```
        int start,end,primecount;
        Primes() {}//default constructor
        Primes(int a, int b) {//Parametrised constructor
            start = a;
            end = b;
            primecount=0;
        }
        void displayPrimes(){
            int is_prime;
            if(start==1){
                start++;//Exception for 1 as it's not a prime
but satisfies the conditions
            }
            System.out.println("\nThe prime numbers in the given
range are:");
            for(int i=start;i<=end;i++) {//Loop for checking
whether a number is prime
                is_prime = 1;
                for (int j = 2; j <= i / 2; j++) {
                    if (i % j == 0) {
                        is_prime = 0;
                        break;
                    }
                }
                if (is_prime == 1) {
                    primecount++;
                    System.out.printf("%d  ", i);//If that
number is prime, print it
                    if(primecount%10==0){//To print only 10
prime numbers in a single line
                        System.out.printf("\n");
                    }
                }
            }
            System.out.println("\nThe number of primes in this
range are "+primecount);
        }

    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);//creating scanner
object to read from stream system.in i.e. user i/p
        System.out.print("Enter range\n");
        int range= sc.nextInt();
        int range2= sc.nextInt();
        System.out.println("The Armstrong numbers in the given
range are:");
        for(int j=range;j<=range2;j++){//checking for all
numbers in the range
            int arms=0,digits=0,temp,n_out;
            int n=j;
            n_out=n;//storing a copy of n(the original number)
for later, as n is changed in the subsequent steps
            temp=n;
            do {
                digits++;
```



```
                temp = temp / 10;
            } while (temp >= 1);//counting number of digits in
each number
            for(int i=0;i<digits;i++){
                temp=n%10;
                arms=arms+(int)Math.pow(temp,digits);
                n=n/10;
            }//calculating the sum of individual digits^total
number of digits

            if(arms==n_out){
                System.out.printf("%d  ",n_out);
            }
        }
        Primes obj=new Primes(range,range2);//creating object of
class Primes nested in class Armstrong
        //As Primes is a static class, we can create an object
without mentioning the parent class
        obj.displayPrimes();//calling class method
    }
}
```

**RESULT:**



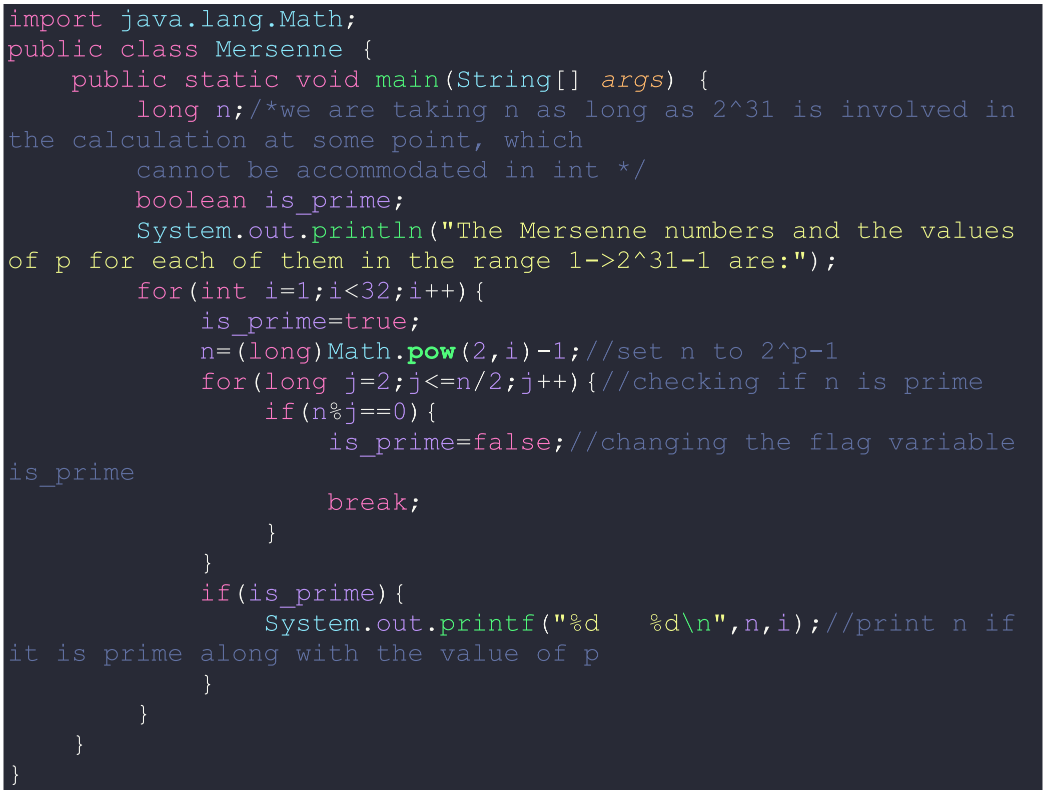
```
Enter range
1 1000
The Armstrong numbers in the given range are:
1  2  3  4  5  6  7  8  9  153  370  371  407
The prime numbers in the given range are:
2  3  5  7  11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97  101  103  107  109  113
127  131  137  139  149  151  157  163  167  173
179  181  191  193  197  199  211  223  227  229
233  239  241  251  257  263  269  271  277  281
283  293  307  311  313  317  331  337  347  349
353  359  367  373  379  383  389  397  401  409
419  421  431  433  439  443  449  457  461  463
467  479  487  491  499  503  509  521  523  541
547  557  563  569  571  577  587  593  599  601
607  613  617  619  631  641  643  647  653  659
661  673  677  683  691  701  709  719  727  733
739  743  751  757  761  769  773  787  797  809
811  821  823  827  829  839  853  857  859  863
877  881  883  887  907  911  919  929  937  941
947  953  967  971  977  983  991  997
The number of primes in this range are 168

Process finished with exit code 0
```

## Program 4

| | |
|---|---|
| **PROBLEM STATEMENT:** | **To find all Mersenne numbers in the range 1-2^31 and print them along with the value of p associated with them.** |
| **THEORY:** | Mersenne primes are a special type of prime number that can be expressed in the form 2^n - 1, where n is any integer. These primes are named after the French mathematician Marin Mersenne, who studied them extensively in the 17th century. Mersenne primes are rare, with only 51 known examples as of 2021. However, they have been the subject of much study in the field of mathematics and have applications in computer science, particularly in the field of cryptography. The largest known prime number as of 2021 is a Mersenne prime with 24,862,048 digits. Despite their rarity, Mersenne primes continue to be an active area of research in number theory, and mathematicians continue to |

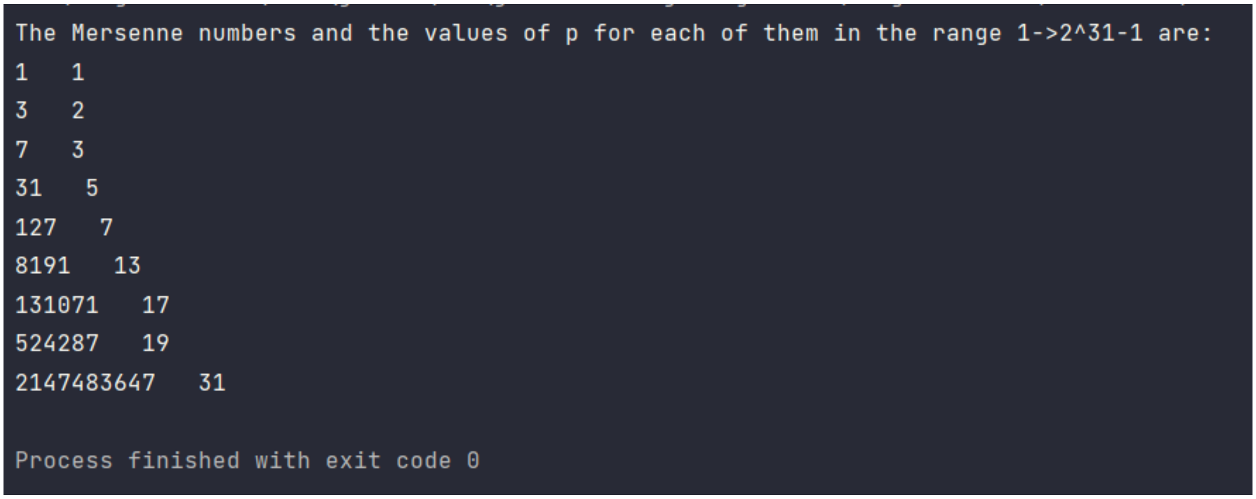
search for larger and more complex examples of these fascinating primes.

**PROGRAM:**



```
import java.lang.Math;
public class Mersenne {
    public static void main(String[] args) {
        long n;/*we are taking n as long as 2^31 is involved in
the calculation at some point, which
        cannot be accommodated in int */
        boolean is_prime;
        System.out.println("The Mersenne numbers and the values
of p for each of them in the range 1->2^31-1 are:");
        for(int i=1;i<32;i++){
            is_prime=true;
            n=(long)Math.pow(2,i)-1;//set n to 2^p-1
            for(long j=2;j<=n/2;j++){//checking if n is prime
                if(n%j==0){
                    is_prime=false;//changing the flag variable
is_prime
                    break;
                }
            }
            if(is_prime){
                System.out.printf("%d    %d\n",n,i);//print n if
it is prime along with the value of p
            }
        }
    }
}
```

**RESULT:**



```
The Mersenne numbers and the values of p for each of them in the range 1->2^31-1 are:
1   1
3   2
7   3
31   5
127   7
8191   13
131071   17
524287   19
2147483647   31

Process finished with exit code 0
```