PRACTICAL PROGRAMS LOGIC (Number-based programs)

Question 1. Hamming Number program

Hamming numbers are positive integer numbers whose prime factors include 2,3 and 5 only
Example:
n=6 is an hamming number as 6=2x3 .So its prime factors are limited to 2 ,3
n=8 is an hamming number as 8=2x2x2 and it has only 2 as its prime factors
n=90 is an hamming number as 90=2x3x3x5 which has only 2,3,5 as prime factors
n=14 is not a hamming number as 14=2x7 .It has 7 as one of its prime factor
n=44 is not a hamming number as 44=2x2x11. It has 11 as one of its prime factors

Design a program to accept any positive integer number and check if it is a Hamming number or not. Display the result with an appropriate message in the format specified below. The program should also generate error message if a negative number is entered.
Test your program for the following data and some random data.

Example 1

```
   INPUT:      Enter any number: 3600
  OUTPUT:       3600= 2 x 2 x 2 x 2 x 3 x 3 x 5 x 5
                3600 IS A HAMMING NUMBER
```

Example 2

```
   INPUT:      Enter any number: 5832
OUTPUT:     5832= 2 x 2 x 2 x 3 x 3 x 3 x 3 x 3 x 3
               5832 IS A HAMMING NUMBER
```

Example 3

```
   INPUT:      Enter any number: 7854
OUTPUT:     7854= 2 x 3 x 7 x 11 x 17
                7854 IS NOT A HAMMING NUMBER
```

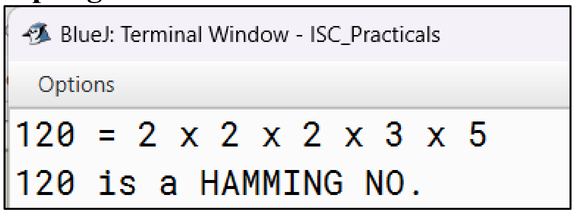
SOLUTION:

```
class Hamming
{
   int n;
   Hamming(int a) //parameterised constructor
   {
      n=Math.abs(a); //storing the input to n, if entered -ve, will be converted to +ve.
   }
   void compute()
   {
      int temp=n, a=2;
      boolean flag=true; //status flag set to true
      System.out.print(n+" = ");
      while(temp>1)
      {
         if(temp%a==0) //if temp (dividend) is divisible by a (divisor)
         {
            temp/=a; //getting the next dividend
            System.out.print(a);
            if(temp>1)
               System.out.print(" x ");
            if(a>5) //if the divisor exceeds 5 status flag is set to false
               flag=false;
         }
         else
         a++; //divisor is increased by 1
      }
      System.out.println();
      if(flag)
```

```
        System.out.println(n+" is a HAMMING NO.");
        else
        System.out.println(n+" is not a HAMMING NO.");
    }
}
```

Output:

Input given - 120



```
120 = 2 x 2 x 2 x 3 x 5
120 is a HAMMING NO.
```

Question 2. Goldbach Number program

A number is said to be a **Goldbach** number, if the number can be expressed as the addition of two odd prime number pairs. If we follow the above condition, then we can find that every even number larger than 4 is a **Goldbach** number because it must have any pair of odd prime number pairs.

Example: 6 = 3,3 (ONE PAIR OF ODD PRIME)
           10 = 3,7 and 5 , 5 (TWO PAIRS OF ODD PRIME)

Write a program to enter any positive EVEN natural number 'N' where (1<=N<=50) and generate odd prime twin of 'N'

Test your program for the following data and some random data.

Example 1

INPUT: N = 14

OUTPUT: ODD PRIME PAIRS ARE: 3, 11
                                    7, 7

Example 2

INPUT: N = 20

OUTPUT: ODD PRIME PAIRS ARE: 17, 3
                                    13, 7

SOLUTION:

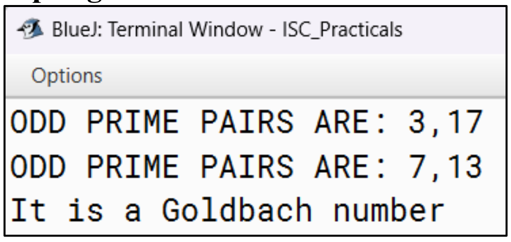
```
class GoldBach
{
   int num;
   int count;

   GoldBach(int a)
   {
      num=Math.abs(a);
      count=0;
   }

   boolean isPrime(int a)
   {
       if(a==1)
         return false;
       for(int i=2; i<=a/2; i++)
       {
            if(a%i==0)
            return false;
        }
        return true;
    }
    void compute()
    {
        count=0;
        for(int i=3;i<=num/2;i++)
        {
            int b=num-i;
            if(isPrime(i) && isPrime(b))
            {
                count++;
                System.out.println("ODD PRIME PAIRS ARE: "+i+","+b);
            }
        }
        if(count>1)
        System.out.println("It is a Goldbach number");
        else
        System.out.println("It is not a Goldbach number");
    }
}
```

Output:

Input given - 20



```
ODD PRIME PAIRS ARE: 3,17
ODD PRIME PAIRS ARE: 7,13
It is a Goldbach number
```

Question 3. Unique-Digit Number/Earth Number program

A unique-digit integer is a positive integer (without leading zeros) with no duplicate digits. For example, 7, 135, 214 are all unique-digit integers whereas 33, 3121, 300 are not.

Given two positive integers *m* and *n*, where *m* < *n*, write a program to determine how many unique-digit integers are there in the range between *m* and *n* (both inclusive) and output them.

The input contains two positive integers *m* and *n*. Assume *m* < *30000* and *n* < *30000*. You are to output the number of unique-digit integers in the specified range along with their values in the format specified below:

Test your program for the following data and some random data.

Example 1

INPUT: m = 100
n = 120

OUTPUT: THE UNIQUE-DIGIT INTEGERS ARE:
102, 103, 104, 105, 106, 107, 108, 109, 120

FREQUENCY OF UNIQUE-DIGIT INTEGERS IS: 9

SOLUTION:

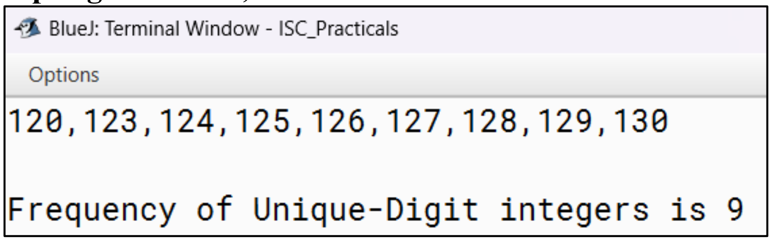
```
class UniqueDGT
{
   int lb, ub;
   UniqueDGT(int a, int b)
   {
      lb=Math.abs(a);
      ub=Math.abs(b);
   }
   boolean checkUnique(int n)
   {
      for(int i=9; i>=0; i--)
      {  int temp=n;
         int count=0;
         while(temp>0)
         {
            if(temp%10==i)
            count++;
            temp/=10;
         }
         if(count>1)
            return false;
      }
      return true;
   }
   void display()
   {
      int i, count=0;
      for(i=lb; i<=ub; i++)
      {
         if(checkUnique(i))
         {  count++;
            System.out.print(i++);
            break;
         }
      }
      for(  ;i<=ub; i++)
      {
         if(checkUnique(i))
         {
            count++;
            System.out.print(","+i);
         }
      }
      System.out.println("\n\nFrequency of Unique-Digit integers is "+count);
   }
}
```

Alternate logic:

```
int num; // it is a global data.
int extractDigit(int x, int y)
{
   if(x==0)
   return 0;
   else
   {
      int r=x%10;
      if(r==y)
         return 1+extractDigit(x/10,y);
      else
         return 0+extractDigit(x/10,y);
   }
}
void check()
{
   boolean flag=true;
   for(int i=1;i<=9;i++)
   {
      int p=extractDigit(num, i);
      if(p>1)
      {
         flag=false;
         break;
      }
   }
   if(flag)
   System.out.println(num + " is a Earth Number");
   else
   System.out.println("Not a Earth Number");
}
```

Output:

Input given - 120, 130



```
120,123,124,125,126,127,128,129,130


Frequency of Unique-Digit integers is 9
```

Question 4. Vampire Number program

A **Vampire** number is a composite natural number with an even number of digits that can be factored into two natural numbers each with half as many digits as the original number and not both with trailing zeros, where the two factors contain precisely all the digits of the original number, in any order of counting multiplicity.

Example: 1260 = 21 x 60 (where, 21 and 60 contain precisely all the digits of the number)

Thus, 1260 is a Vampire number.

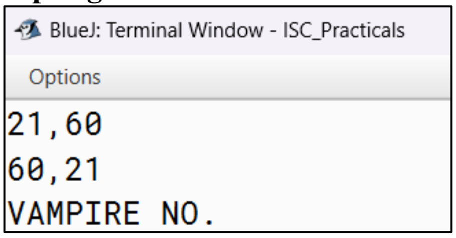
Accept two positive integers m and n, where m is less than n and the values of both 'm' and 'n' must be greater than or equal to 1000 and less than or equal to 9999 as user input. Display all Vampire numbers that are in the range between m and n (both inclusive) and output them along with the frequency, in the format specified below:

Some 4-digit Vampire numbers are 1260, 1395, 1435, 1530, 1827, 2187, and 6880.

SOLUTION:

```
class Vampire
{
   int num;
   Vampire(int a)
   {
      num=Math.abs(a);
   }
   int digitCount(int n)
   {
      int c=0;
      for(int a=n;a>0;a=a/10)
         c++;
      return c;
   }
   int sortDigit(int a)
   {
      int temp=0;
      for(int i=9;i>=0;i--)
      {
         for(int j=a;j>0;j=j/10)
         {
            int k=j%10;
            if(k==i)
            temp=temp*10+k;
         }
      }
      return temp;
   }
   boolean compute()
   {
      boolean flag=false;
      int c=digitCount(num)/2;
      int n1=sortDigit(num);
      int lb=(int)Math.pow(10,c-1);
      int ub=(int)Math.pow(10,c)-1;
      for(int i=lb;i<ub;i++)
      {
         if(num%i==0)
         {
            int j=num/i;
            if(digitCount(j)==digitCount(i))
            {
               int p=i*(int)Math.pow(10,c)+j;
               if(sortDigit(p)==sortDigit(num))
               {
                  System.out.println(i+","+j);
                  flag=true;
               }
            }
         }
      }
      return flag;
   }
   void checkVampire()
   {
      if(compute())
      System.out.println("VAMPIRE NO.");
      else
      System.out.println("Not a Vampire no.");
   }
}
```

Output:
Input given - 1260



```
21,60
60,21
VAMPIRE NO.
```

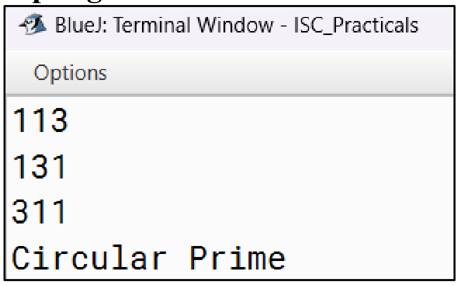
Question 5. Circular Prime Number program
Circular Prime number is a number more than one digit. Any number formed by shifting the digits in the number to their positions either left or right is a prime number. For example, 113 is a prime number. Now, if we shift the digits to their left side, we will get two new numbers—131 and 311—and both these numbers are prime numbers.

SOLUTION:

```
class CircularPrime
{
   int n;
   CircularPrime(int a)
   {
      n=Math.abs(a);
   }

   int countlen(int a)
   {
      if(a==0)
      return 0;
      else if(a>=1&&a<=9)
      return 1;
      else
      return 1+countlen(a/10);
   }
```

```
    boolean primeCheck(int a)
    {
       if(a==1)
       return false;
       else if(a==2||a==3||a==5||a==7)
       return true;
       else if(a%2==0||a%3==0||a%5==0||a%7==0)
       return false;
       else
       return true;
    }
    void circularNumber()
    {
       int p=countlen(n);
       int cn=n, c=0;
       for(int i=1; i<=p; i++)
       {
          System.out.println(cn);
          if(primeCheck(cn))
             c++;
          int r=(int)Math.pow(10,p-1);
          cn=n%r*10+n/r;
          n=cn;
       }
       if(c==p)
       System.out.println("Circular Prime");
       else
       System.out.println("Not a Circular Prime");
    }
}
```
Output:
Input given - 113



```
113
131
311
Circular Prime
```

Question 6. Auto Biographical Number program

An Autobiographical number is number where the number N has the first digit represents the total count of zeroes (0) in N, the second digit represents the total count of ones (1) in N, the third digit represents the total count of twos (2) and so on.

For example, consider the Autobiographical numbers given below:

1210 : 1 Zero, 2 Ones, 1 Twos and 0 Threes.

21200 : 2 Zero, 1 Ones, 2 Twos, 0 Threes and 0 Fours.

Write a program in Java to accept one positive number and check and print it is an Autobiographical number or not. The program should give the output in the format given below.

Test your program with the following data:

Example 1:

INPUT: 21200

OUTPUT: It is an Autobiographical number.

2 Zero 1 One 2 Two 0 Three 0 Four

Example 2:

INPUT: 3211000

OUTPUT: It is an Autobiographical number.

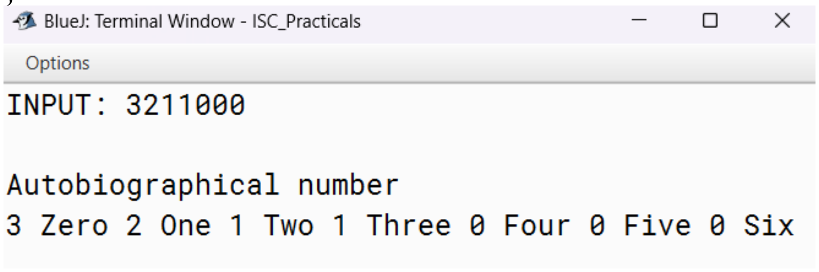
3 Zero 2 One 1 Two 1 Three 0 Four 0 Five 0 Six

SOLUTION:

```
class AutoBioNo
{
   long n, ln;
   int ar[], count[];
   AutoBioNo(long a)
   {
      n=Math.abs(a);
      ln=(int)Long.toString(n).length();
      ar=new int[ln];
      count=new int[10];
   }
   private void compute()
   {
      int i=ln-1;
      for(long a=n; a>0; a=a/10)
      {
         ar[i--]=(int)(a%10);
      }
      for(i=0; i<10; i++)
      {
         for(long a=n; a>0; a=a/10)
         {
            int r=(int)(a%10);
            if(r==i)
               count[i]++;
         }
      }
   }
```

```
    private boolean checkAutoBio()
    {
        compute();
        for(int i=0; i<ln; i++)
        {
            if(ar[i]!=count[i])
                return false;
        }
        return true;
    }

    public void display()
    {
        System.out.println("INPUT: "+n);
        if(checkAutoBio())
        {
            System.out.println("\nAutobiographical number");
            for(int i=0; i<ln; i++)
            {
                System.out.print(count[i]+" ");
                switch(i)
                {
                    case 0:System.out.print("Zero ");
                    break;
                    case 1:System.out.print("One ");
                    break;
                    case 2:System.out.print("Two ");
                    break;
                    case 3:System.out.print("Three ");
                    break;
                    case 4:System.out.print("Four ");
                    break;
                    case 5:System.out.print("Five ");
                    break;
                    case 6:System.out.print("Six ");
                    break;
                    case 7:System.out.print("Seven ");
                    break;
                    case 8:System.out.print("Eight ");
                    break;
                    case 9:System.out.print("Nine ");
                    break;
                }
            }
        }
        else
            System.out.println("\nNot Autobiographical number");
    }
}
```



```
INPUT: 3211000

Autobiographical number
3 Zero 2 One 1 Two 1 Three 0 Four 0 Five 0 Six
```

Question 7. Scary Number program

A **Scary** number is any positive integer that contains 13 at any position. For example, 113, 131, 1013, 1300 etc. But, if the number contains multiple 13, it will be no longer a Scary Number. For example 1313, 13013, 131313 etc. Design a class that will accept two numbers as Lower bound and Upper bound and display all the scary numbers in between. If there is no Scary number, it will print appropriate message. The program should validate the input with proper coding.

Example 1:

INPUT: 10 30

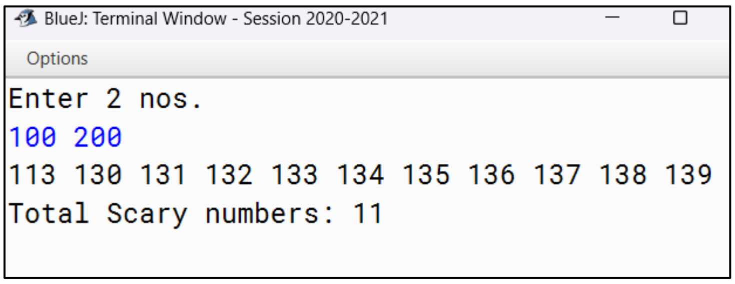
OUTPUT: 13

TOTAL SCARY NUIMBERS ARE : 1

SOLUTION:

```
import java.util.Scanner;
class ScaryNumber
{
   int a, b; // data members
   public ScaryNumber(int i, int j) //parameterized constructor
   {
      a=Math.abs(i); b=Math.abs(j);
   }
   public boolean check(int n)
   {
      int r, c=0;
      for( ; n>0; n/n/10)
      {
         r=n%100;
         if(r==13) //checking for the no. 13
            c++; //incrementing the counter
      }
      if(c==1) //if only one 13 is present
      return true;
      else //more than one 13 is present
      return false;
   }
   public void display()
   {
      if(a>b)
      {
         a=a+b;
         b=a-b;
         a=a-b;
      }
      int c=0, d=0;
      for(int i=a; i<=b; i++)
      {
         if(check(i))//calling check() function
         {
            System.out.print(i+" ");
            c++;
            if(c%22==0)//for setting tabular printing
              System.out.println();
         }
      }
      System.out.println("\nTotal Scary numbers: "+c);
   }
```

```
    public static void main(String ar[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter 2 nos.");
        int i=sc.nextInt();
        int j=sc.nextInt();
        ScaryNumber sobj=new ScaryNumber(i,j); //creating the object
        sobj.display();
    }
}
```



```
Enter 2 nos.
100 200
113 130 131 132 133 134 135 136 137 138 139
Total Scary numbers: 11
```

Some more number-based programs for PRACTICAL EXAM
Students need to build the class structure per the question given and add member data and member methods accordingly. Here, for their help, only the main logic is provided.

Fascinating No. - When a number (3 digits or more) is multiplied by 2 and 3, and when both these products are concatenated with the original number, then it results in all digits from 1 to 9 present exactly once.
E.g. 192
192*1=192
192*2=384
192*3=576
192382576
Solution logic:

```
        System.out.println("Enter a Number:-");
        int a=sc.nextInt();
        int n, m=0, p=0;
        int b=a*2, c=a*3;
        String s = Integer.toString(a)+Integer.toString(b)+Integer.toString(c);
        int x=Integer.parseInt(s);
        while(x!=0)
        {
            n=x%10;
            m=m+n;
            p++;
            n=n/10;
        }
        if(m==45 && p==9)
        System.out.println("It's a fascinating Number");
        else
        System.out.println("It's not a fascinating Number");
```

Decimal to Hexadecimal conversion of a number

```
    public void convert(int n)//method for compute
    {
        if(n==0)
        return;
        else
        {
           int a=n%16;//remainder of the division
           if(a>=0&&a<=9)//remainder is a digit within 1 to 9
                     hexa+=""+a;
           else
           {   switch(a)
              {
              case 10:hexa+='A'; break;
              case 11:hexa+='B'; break;
              case 12:hexa+='C'; break;
              case 13:hexa+='D'; break;
              case 14:hexa+='E'; break;
              case 15:hexa+='F'; break;
              }
           }
           convert(n/16);//next number i.e. divisor
        }
     }
```

To form the smallest number with the digits of a given number, where all the digits will be used in the number itself. For example, if the input is 10203, the smallest number formed will be 10023 and not 00123 because 00123 is 123, and zeros will have no value at the beginning.

Solution of the main logic:

```
   public void compute(int num)
   {
      int c=0, d=0, t=num;
      while(t>0)
      {
         if(t%10==0)
            c++;
         t=t/10;
      }
      for(int i=1; i<=9; i++)
      {
         t=num;
         while(t>0)
         {
            if(t%10==i)
            {
               res=res*10+t%10;
               d++;
            }
            t=t/10;
         }
      }
      int p=(int)Math.pow(10,c);
      int a=res%(int)Math.pow(10,d-1);
      int b=res/(int)Math.pow(10,d-1);
      res=b*(int)Math.pow(10,c+d-1)+a;
   }
```

~Thank You~