Problem No. 1
We use the integers a, b, and n to create the following series:
(a+2^0 *b),(a+2^0 *b+2^1*b) , (a+2^0 *b+2^1*b +2^2*b), ……
(a+2^0 *b+2^1*b+2^2*b+....+2^n-1 * b)
You are given a, b, and n . For given a, b, and n print the series of numbers.
Constraint:
0<=a,b<=50
0<=n<=15
Output Format
print the corresponding series on a new line.
Sample Input
5 3 5
Sample Output
8 14 26 50 98

```
import java.util.Scanner;

public class NumberSeries {

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

       System.out.print("Enter a: ");
       int a = scanner.nextInt();

       System.out.print("Enter b: ");
       int b = scanner.nextInt();

       System.out.print("Enter n: ");
       int n = scanner.nextInt();

       if (a < 0 || a > 50 || b < 0 || b > 50 || n < 0 || n > 15) {
           System.out.println("Invalid input. Please ensure 0 <= a, b <= 50 and
0 <= n <= 15.");
           return;        }

       double sum = a;
       System.out.print(sum + " ");
       for (int i = 1; i <= n; i++) {
           sum += b * Math.pow(2, i - 1);
           System.out.print((long) sum + " ");
       }
```

```
        System.out.println(); // Add newline for clarity
    }
}
```

Problem No: 02
Given integer x. Write a code to print its reverse.
ReverseOfx(123) → 321
ReverseOfx(-123) → -321
ReverseOfx(406) → 604.

```
import java.util.Scanner;

public class ReverseInteger {

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

        System.out.print("Enter an integer: ");
        int x = scanner.nextInt();

        int reversed = reverse(x);
        System.out.println("Reverse of " + x + " is " + reversed);
    }

    public static int reverse(int x) {
        int reversed = 0;
        boolean isNegative = x < 0;

        if (isNegative) {
            x = -x;
        }

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

        return isNegative ? -reversed : reversed;
    }
}
```

Problem No: 03
Given positive integer x. Write a program to print a Binary number of x.
BinaryNum(23) → 10111
BinaryNum(124) → 1111100
BinaryNum(234) → 11101010

```
import java.util.Scanner;

public class BinaryNumber {

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

       System.out.print("Enter a positive integer: ");
       int x = scanner.nextInt();

       if (x <= 0) {
           System.out.println("Invalid input. Please enter a positive
integer.");
           return;
       }

       String binary = "";
       while (x > 0) {
           binary = (x % 2) + binary;
x /= 2;
       }

       System.out.println("Binary representation of " + x + " is " + binary);
   }
}
```

Problem No: 04
Write a program to implement the following Bus Ticket scenario.
Read From stage number and To stage number.
Read the number of adult and children passengers.
Calculate the number of stages they are traveling.
Calculate adult cost @ Rs.10 per passenger per stage.
Calculate child cost @ Rs.5 per passenger per stage.
Find total ticket cost.
Find the discount of the ticket as follows:
If adults>=5 calculate a discount of 20% on ticket cost.
else If adults==4 calculate a discount of 15% on ticket cost.
else If adults==3 calculate a discount of 10% on ticket cost.
else If adults==2 calculate a discount of 5% on ticket cost.
else calculate a discount of 0% on ticket cost.
Then find the ticket cost after discount.
And also find the service charge of 5% on ticket cost.
Find the total ticket cost and display the ticket cost.

```
import java.util.Scanner;

public class BusTicket {

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

       System.out.print("Enter From stage number: ");
       int fromStage = scanner.nextInt();
       System.out.print("Enter To stage number: ");
       int toStage = scanner.nextInt();


       int stagesTraveled = Math.abs(toStage - fromStage);


       System.out.print("Enter number of adult passengers: ");
       int adultPassengers = scanner.nextInt();
       System.out.print("Enter number of children passengers: ");
       int childPassengers = scanner.nextInt();


       double adultCost = stagesTraveled * 10.0;
       double childCost = stagesTraveled * 5.0;
```

```
        double totalCost = adultPassengers * adultCost + childPassengers *
childCost;

        double discountPercentage = 0.0;
        if (adultPassengers >= 5) {
            discountPercentage = 0.2;
        } else if (adultPassengers == 4) {
            discountPercentage = 0.15;
        } else if (adultPassengers == 3) {
            discountPercentage = 0.1;
        } else if (adultPassengers == 2) {
            discountPercentage = 0.05;
        }

        double discountAmount = discountPercentage * totalCost;


        double costAfterDiscount = totalCost - discountAmount;

        double serviceCharge = costAfterDiscount * 0.05;

        double finalCost = costAfterDiscount + serviceCharge;

        System.out.println("From stage: " + fromStage);
        System.out.println("To stage: " + toStage);
        System.out.println("Number of stages traveled: " + stagesTraveled);
        System.out.println("Number of adult passengers: " + adultPassengers);
        System.out.println("Number of children passengers: " + childPassengers);
        System.out.printf("Adult passenger cost: ₹%.2f\n", adultCost);
        System.out.printf("Child passenger cost: ₹%.2f\n", childCost);
        System.out.printf("Total cost before discount: ₹%.2f\n", totalCost);
        System.out.printf("Discount (%.0f%%): ₹%.2f\n", discountPercentage *
100, discountAmount);
        System.out.printf("Cost after discount: ₹%.2f\n", costAfterDiscount);
        System.out.printf("Service charge (5%%): ₹%.2f\n", serviceCharge);
        System.out.printf("Final ticket cost: ₹%.2f\n", finalCost);
    }
}
```

Problem No: 5.

Given positive integer X. print the nearest prime number to the given X.
NearesPrime(11) → 11
NearestPrime(25) → 23
NearestPrime(21) → 19 23
NearestPrime(6) → 5 7

```
import java.util.Scanner;

public class NearestPrime {

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

       System.out.print("Enter a positive integer: ");
       int x = scanner.nextInt();

       int[] nearestPrimes = nearestPrime(x);

       System.out.println("Nearest prime numbers to " + x + ": " +
nearestPrimes[0] + " " + nearestPrimes[1]);
   }

   public static int[] nearestPrime(int x) {

       if (x <= 1) {
           return new int[] {-1, -1};
       }

       if (x % 2 == 0) {
           if (x == 2) {
               return new int[] {2, 3};
           } else {
               return new int[] {2, x - 1};
           }
       } else {

           int i = x + 2;
           while (true) {
               boolean isPrime = true;
               for (int j = 3; j <= Math.sqrt(i); j += 2) {
                   if (i % j == 0) {
                       isPrime = false;
                       break;
                   }
```

```
                }
                if (isPrime) {
                    return new int[] {x, i};
                } else {
                    i += 2;
                }
            }
        }
    }
}
```

Problem No. 6

Given positive integer X. Find the sum of prime digits of X is a Prime or not. Return true if it is a prime number. Else return false.

PrimeDigitSum(1234) —> true [2+3 = 5]

PrimeDigitSum(5677) → true [2+7+7 = 19]

PrimeDigitSum(987) —> true [7 = 7]

PrimeDigitSum(3456) → false [3+5 = 8 is not a prime]

```
public class PrimeDigitSum {

    public static boolean isPrime(int num) {
            if (num <= 1) {
            return false;
        }
        if (num <= 3) {
            return true;
        }
        if (num % 2 == 0 || num % 3 == 0) {
            return false;
        }
        int i = 5;
        while (i * i <= num) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
            i += 6;
        }
        return true;
    }

    public static boolean primeDigitSumIsPrime(int num) {
            int sumOfPrimes = 0;
```

```
        while (num > 0) {
            int digit = num % 10;
            if (isPrime(digit)) {
                sumOfPrimes += digit;
            }
            num /= 10;
        }
        return isPrime(sumOfPrimes);
    }

    public static void main(String[] args) {
        int number = 1234;
        if (primeDigitSumIsPrime(number)) {
            System.out.println(number + " is a prime number with a prime sum of
digits (" + primeDigitSum(number) + ")");
        } else {
            System.out.println(number + " is not a prime number with a prime sum
of digits (" + primeDigitSum(number) + ")");
        }
    }
}
```

Problem No. 7

Given positive integer X. Print the nearest Armstrong number of given X.

NearestArmstrong(5) → 5

NearestArmstrong(99) → 153

NearestArmstrong(450) → 407

NearestArmstrong(1600) → 1634

```
import java.util.Scanner;

public class NearestArmstrong {

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

        System.out.print("Enter a positive integer: ");
        int x = scanner.nextInt();

        int nearestArmstrong = nearestArmstrongNumber(x);
```

```
        System.out.println("Nearest Armstrong number to " + x + ": " +
nearestArmstrong);
    }

    public static int nearestArmstrongNumber(int x) {
        if (isArmstrongNumber(x)) {
            return x;
        }

        int lowerArmstrong = 0;
        int upperArmstrong = 0;
        int i = x - 1;
        while (lowerArmstrong == 0 || upperArmstrong == 0) {
            if (isArmstrongNumber(i)) {
                lowerArmstrong = i;
            }
            if (isArmstrongNumber(x + i)) {
                upperArmstrong = x + i;
            }
            i--;
        }


        return Math.abs(x - lowerArmstrong) < Math.abs(x - upperArmstrong) ?
lowerArmstrong : upperArmstrong;
    }

    public static boolean isArmstrongNumber(int num) {
        int originalNum = num;
        int sum = 0;
        int power = countDigits(num);
        while (num > 0) {
            int digit = num % 10;
            sum += Math.pow(digit, power);
            num /= 10;
        }
        return sum == originalNum;
    }

    public static int countDigits(int num) {
        int count = 0;
        while (num > 0) {
            num /= 10;
            count++;
        }
        return count;
    }
}
```

Problem No. 8
Given two positive integers X and Y that indicate a range of numbers. Print the number from the same range which is a fibonacci term and a prime too. If not print 0.
FibPrime(2,25) → 2 3 5 13
FibPrime(1,100) → 2 3 5 13 89
FibPrime(25, 75) → 0

```
import java.util.ArrayList;

public class FibPrime {

   public static void main(String[] args) {
       int x = 2;
       int y = 25;

       ArrayList<Integer> fibPrimes = fibPrime(x, y);

       if (fibPrimes.isEmpty()) {
           System.out.println("No Fibonacci numbers that are also prime within
the range.");
       } else {
           System.out.println("Fibonacci numbers that are also prime within the
range: " + fibPrimes);
       }
   }

   public static ArrayList<Integer> fibPrime(int x, int y) {
       ArrayList<Integer> fibPrimes = new ArrayList<>();

       int a = 0, b = 1;
       int next = a + b;

       while (next <= y) {
           if (next >= x && isPrime(next)) {
               fibPrimes.add(next);
           }
           a = b;
           b = next;
           next = a + b;
       }

       return fibPrimes;
```

```
    }

    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        if (num <= 3) {
            return true;
        }
        if (num % 2 == 0 || num % 3 == 0) {
            return false;
        }
        int i = 5;
        while (i * i <= num) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
            i += 6;
        }
        return true;
    }
}
```

Problem No. 9

Given positive integer X. Check if X is a fibonacci term or not. If it is a fibonacci term then check if it is a Prime number or not. If it is a Prime number too, then print X as result. Else print Nearest Fibonacci Prime number of given X.

FibPrime(29) → 13

FibPrime(79) → 89

FebPrime(13) → 13

```
public class FibPrime {

    public static boolean isPrime(int num) {
                if (num <= 1) {
            return false;
        }
        if (num <= 3) {
            return true;
        }
        if (num % 2 == 0 || num % 3 == 0) {
            return false;
```

```
        }
        int i = 5;
        while (i * i <= num) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
            i += 6;
        }
        return true;
    }

    public static boolean isFibonacci(int num) {
           if (num == 0 || num == 1) {
            return true;
        }
        int a = 0, b = 1;
        while (b < num) {
            int temp = a + b;
            a = b;
            b = temp;
        }
        return b == num;
    }

    public static int nearestFibPrime(int x) {

        if (isFibonacci(x) && isPrime(x)) {
            return x;
        }

        int a = 0, b = 1;
        while (true) {
            if (isPrime(b)) {
                return b;
            }
            int temp = a + b;
            a = b;
            b = temp;
        }
    }

    public static void main(String[] args) {
        int x = 29;
        int nearestPrime = nearestFibPrime(x);
        System.out.println(nearestPrime);
    }
}
```