OOP EXPERIMENT-4

NAME-ANMOL

SAP-590011794

BATCH-20

DATE-22 FEB 2026

SUBMITTED TO- PROF. Kalluri Shareef Babu

Theory

Bitwise Operators

Operators used: &, |, ^, ~, <<, >>.

& is used for carry generation in bitwise addition and for power of 2 checking logic.

^ is used to add bits without carry in bitwise addition.

<< is used to shift carry in bit-level addition.

|, ^, ~, <<, >> are demonstrated on integer inputs for bit manipulation.

Relational and Logical Checks

Relational operators (>,<,==) are used in condition checks.

Logical OR (||) is used in overflow boundary validation.

Increment and Decrement Operators

Post and pre forms are used: a++, ++a,a--,--a.

Demonstrates evaluation order difference between post and pre operations.

Type Conversion and Casting

Implicit conversion: int to double

Explicit conversion: double to int using cast

Explicit cast dose'nt care about decimal part turns it to 0 (precision loss).

Data Type Limits and Overflow Check

Integer.MAX_VALUE and Integer.MIN_VALUE are used to check safe multiplication by 2 range.

Boundary comparison avoids overflow before performing multiplication.

Math Functions

Math.pow(base, exponent) is used for compound interest computation.

Used with floating point arithmetic for formula evaluation.

Input Handling

Scanner is used for keyboard integer input in most exercise files.

nextInt() reads numeric input values.

Command-line arguments are also used through args[].

Integer.parseInt(args[i]) converts argument text into integers.

Output Functions

System.out.print() for inline output.

System.out.println() for line-based output.

Loops

while loop for repeated checks and iterative processing.

do-while loop for post-condition iteration.

for loop for controlled counter-based iteration.

Nested loops are used for 2D and 3D array traversal.

Recursion is used to compute digit sum without loops.

Conditional Branching

if-else is used for decisions like overflow, power of 2, and prime check.

switch-case is used for menu-style selection flow.

break is used in loops and switch to stop control flow when required.

Arrays

2D arrays are declared and traversed using nested loops.

3D arrays are declared and traversed using triple nested loops.

Index based access is used to print matrix/cube elements.

OOP and Java-Specific Features

Class and object creation is used to organize logic into methods.

Method calls encapsulate operations such as add, swap, evaluate, and compute.

Inheritance is demonstrated with subclass extending superclass.

instanceof operator is used for runtime type relationship checking.

Number Property Logic

Power of 2 check uses bit rule: n & (n-1).

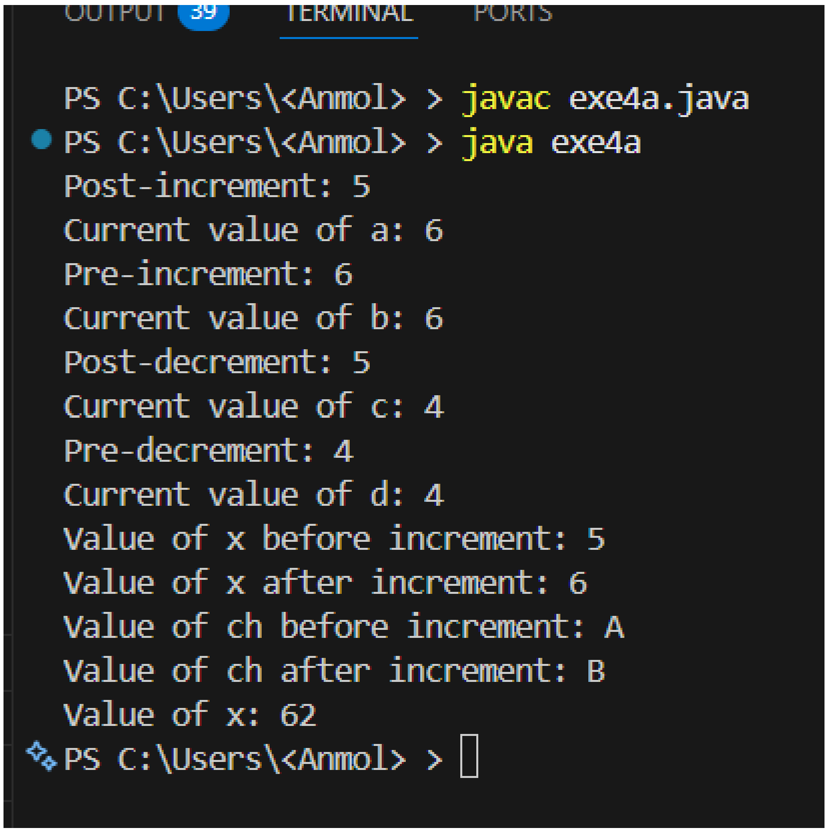
Prime validation uses trial division loop and early exit with break.

Digit-based logic uses repeated modulo and division operations.

Classwork

Exe4a

```java
//program to show post and pre increment and decrement operator
public class exe4a
{
    public static void main(String[] args) {
        int a=5;
        System.out.println("Post-increment: "+(a++));
        System.out.println("Current value of a: "+a);
        int b=5;
        System.out.println("Pre-increment: "+(++b));
        System.out.println("Current value of b: "+b);
        int c=5;
        System.out.println("Post-decrement: "+(c--));
        System.out.println("Current value of c: "+c);
        int d=5;
        System.out.println("Pre-decrement: "+(--d));
        System.out.println("Current value of d: "+d);
        int x=5;
        System.out.println("Value of x before increment: "+x);
        x++;
        System.out.println("Value of x after increment: "+x);
        char ch='A';
        System.out.println("Value of ch before increment: "+ch);
        ch++;
        System.out.println("Value of ch after increment: "+ch);
        x=a+b+10*5;
        System.out.println("Value of x: "+x);
    }
}
```

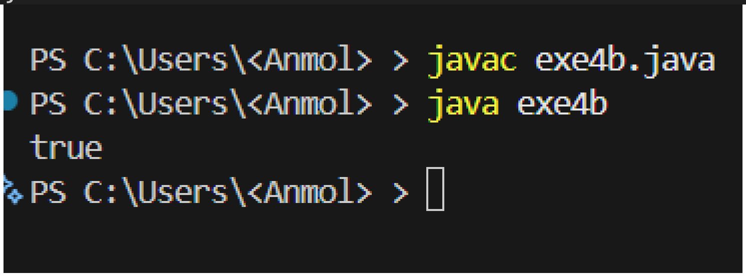


```
PS C:\Users\<Anmol> > javac exe4a.java
PS C:\Users\<Anmol> > java exe4a
Post-increment: 5
Current value of a: 6
Pre-increment: 6
Current value of b: 6
Post-decrement: 5
Current value of c: 4
Pre-decrement: 4
Current value of d: 4
Value of x before increment: 5
Value of x after increment: 6
Value of ch before increment: A
Value of ch after increment: B
Value of x: 62
PS C:\Users\<Anmol> >
```

Observation: The program demonstrates the use of post-increment, pre-increment, post-decrement, and pre-decrement operators. It shows how the value of the variable changes before and after the increment/decrement operation. Additionally, it evaluates an expression involving addition and multiplication to show operator precedence in Java.

Exe4b

```java
class Msc
{
    String name;
}
class Mscam extends Msc
{
    String name;
}
public class exe4b
{
public static void main(String args[])
{
    boolean v;
    Msc st= new Msc();
    Mscam st1= new Mscam();
    v=st1 instanceof Msc;
    System.out.println(v);
}
}
```

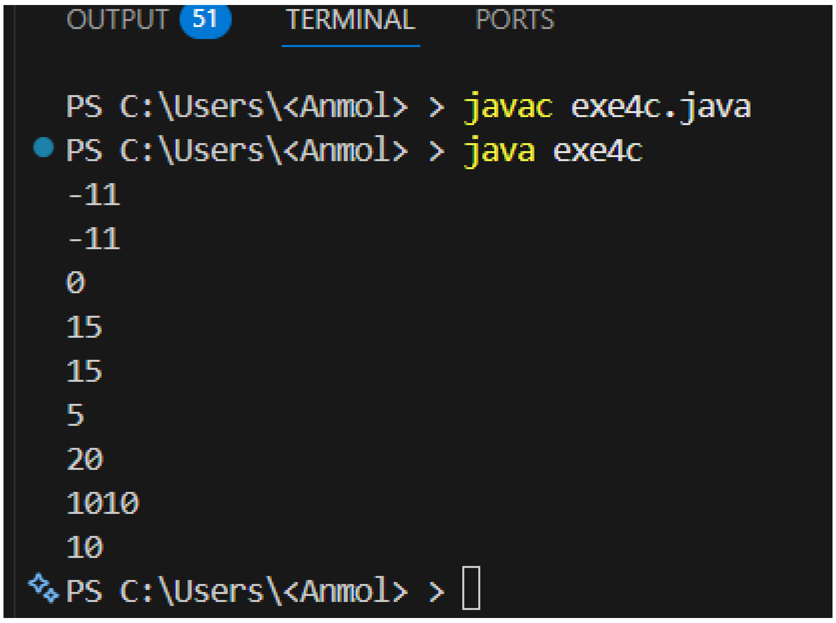


```
PS C:\Users\<Anmol> > javac exe4b.java
PS C:\Users\<Anmol> > java exe4b
true
PS C:\Users\<Anmol> > 
```

Observation: The program demonstrates the use of the 'instanceof' operator in Java. It checks if an object of the Mscam class is an instance of the Msc class, which is true since Mscam is a subclass of Msc. The output will be 'true' indicating that the object st1 is indeed an instance of the Msc class.

Exe4c

```
class exe4c
{
    public static void main(String[] args)
    {
        int num=10;
        int num1=0b1010; //binary literal
        System.err.println(~num);
        System.err.println(~num1);
        int a=10;
        int b=5;
        System.out.println(a&b);
        System.out.println(a|b);
        System.out.println(a^b);
        System.out.println(a>>1);
        System.out.println(a<<1);
        int c=1010;
        int d=1011;
        int e=0b1010;
        int f=0b1011;
        System.out.println(c&d);
        System.out.println(e&f);
    }
}
```

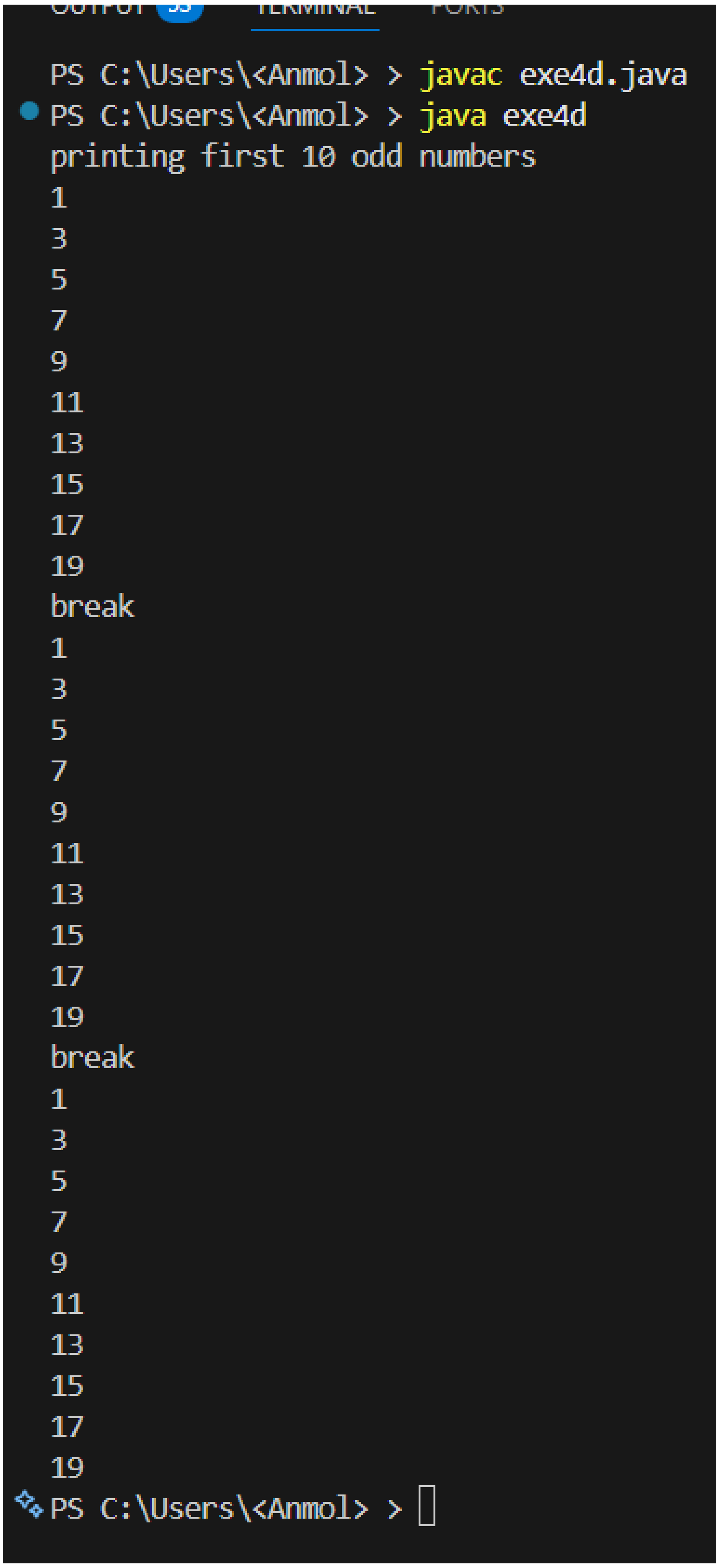


```
PS C:\Users\<Anmol> > javac exe4c.java
PS C:\Users\<Anmol> > java exe4c
-11
-11
0
15
15
5
20
1010
10
PS C:\Users\<Anmol> >
```

Observation: The program uses bitwise operations (&, |, ^, >>, <<) to perform various operations on integers. The output is displayed to the user as specified in the print statements and the program demonstrates the use of binary literals in Java.

Exe4d

```
public class exe4d
{
    public static void main(String[] args)
    {
        int c=1;
        System.out.println("printing first 10 odd numbers");
        while(c<=20)
        {
            System.out.println(c);
            c=c+2;
        }
        c=1;
        System.out.println("break");
        do
        {
            System.out.println(c);
            c=c+2;
        }
        while(c<20);
        c=1;
        System.out.println("break");
        for(int i=1;i<=20;i=i+2)
        {
            System.out.println(i);
        }
    }
}
```



```
PS C:\Users\<Anmol> > javac exe4d.java
PS C:\Users\<Anmol> > java exe4d
printing first 10 odd numbers
1
3
5
7
9
11
13
15
17
19
break
1
3
5
7
9
11
13
15
17
19
break
1
3
5
7
9
11
13
15
17
19
PS C:\Users\<Anmol> >
```

Observation: The program demonstrates the use of while, do-while, and for loops to print the first 10 odd numbers. Each loop iterates through the odd numbers by incrementing the counter by 2. The output will show the odd numbers from 1 to 19, with a break between each loop to separate the outputs for clarity.

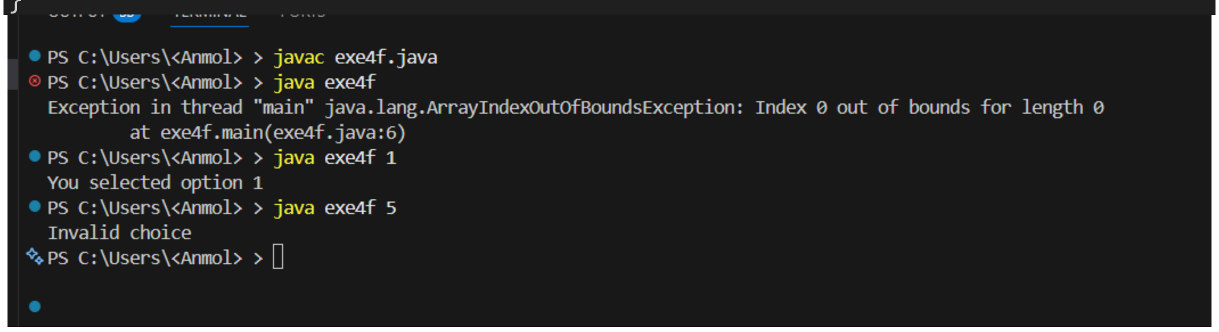
Exe4e

```
//a loop program to find prime number only using loops
public class exe4e
{
    public static void main(String[] args)
    {
        int num=Integer.parseInt(args[0]);
        boolean isPrime=true;
        for(int i=2;i<=num/2;i++)
        {
            if(num%i==0)
            {
                isPrime=false;
                break;
            }
        }
        if(isPrime)
        {
            System.out.println(num+" is a prime number.");
        }
        else
        {
            System.out.println(num+" is not a prime number.");
        }
    }
}
```

Observation: The program checks if a given number is prime by using a for loop to test divisibility from 2 up to half of the number. If the number is divisible by any of these values, it is not prime. The program then prints whether the number is prime or not based on the result of the check.

Exe4f

```java
// menu selection through switch case without libraries
class exe4f
{
    public static void main(String[] args)
    {
        int choice=Integer.parseInt(args[0]);
        switch(choice)
        {
            case 1:
            System.out.println("You selected option 1");
            break;
            case 2:
            System.out.println("You selected option 2");
            break;
            case 3:
            System.out.println("You selected option 3");
            break;
            default:
            System.out.println("Invalid choice");
        }
    }
}
```

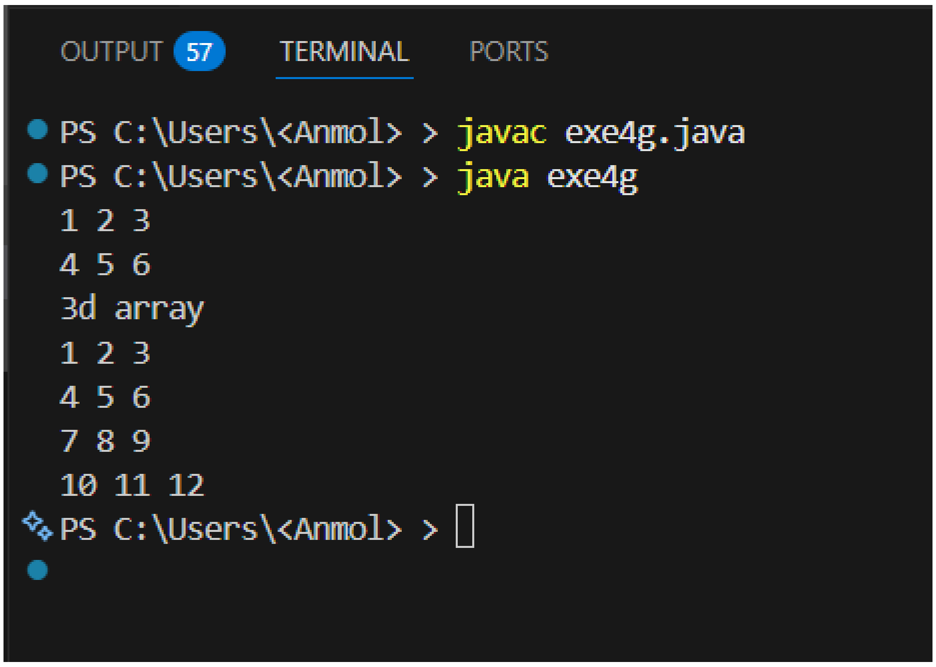


```
PS C:\Users\<Anmol> > javac exe4f.java
PS C:\Users\<Anmol> > java exe4f
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
        at exe4f.main(exe4f.java:6)
PS C:\Users\<Anmol> > java exe4f 1
You selected option 1
PS C:\Users\<Anmol> > java exe4f 5
Invalid choice
PS C:\Users\<Anmol> >
```

Observation: The program demonstrates the use of a switch-case statement to handle menu selection based on user input. The user is expected to provide a choice as a command-line argument, which is then parsed into an integer. The switch statement evaluates the choice and executes the corresponding case block, printing a message for each valid option (1, 2, or 3). If the user enters an invalid choice, the default case is executed, printing "Invalid choice".

Exe4g

```java
//2d arary and 3d array without libraries
class exe4g
{
    public static void main(String[] args)
    {
        int a[][]={
        {1,2,3},
        {4,5,6}
        };
        for(int i=0;i<2;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
        System.out.println("3d array");
    int b[][][]={
        {
            {1,2,3},
            {4,5,6}
        },
        {
            {7,8,9},
            {10,11,12}
        }
    };
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            for(int k=0;k<3;k++)
            {
                System.out.print(b[i][j][k]+" ");
            }
            System.out.println("  ");
        }
    }
    }
}
```



```
PS C:\Users\<Anmol> > javac exe4g.java
PS C:\Users\<Anmol> > java exe4g
1 2 3
4 5 6
3d array
1 2 3
4 5 6
7 8 9
10 11 12
PS C:\Users\<Anmol> >
```

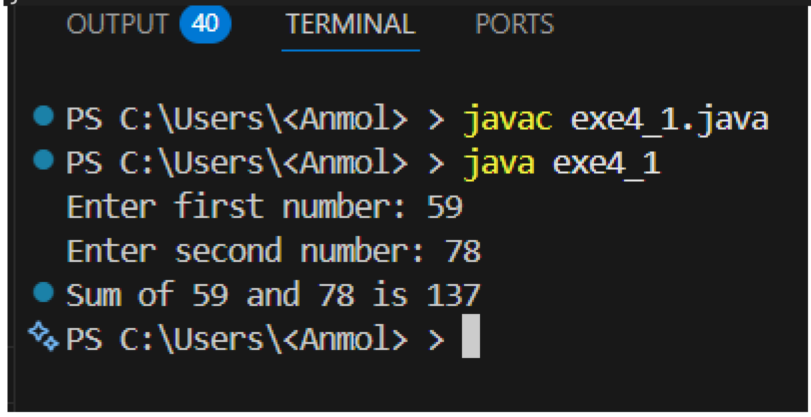
Observation: The program demonstrates the use of 2D and 3D arrays in Java without using any libraries. It initializes a 2D array 'a' with 2 rows and 3 columns, and a 3D array 'b' with 2 blocks, each containing 2 rows and 3 columns. The program then uses nested loops to iterate through the elements of both arrays and prints them in a formatted manner. The output will display the elements of the 2D array followed by the elements of the 3D array in a structured format.

Homework

Exe4_1

```
//Without using arithmetic operators (+, -,*,/), write a program to add two
integers.
import java.util.Scanner;
class add
{
    int add(int a,int b)
    {
        while(b!=0)
        {
            int carry=a&b;
            a=a^b;
            b=carry<<1;
        }
        return a;
    }
```

```java
}
public class exe4_1
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter first number: ");
        int num1=sc.nextInt();
        System.out.print("Enter second number: ");
        int num2=sc.nextInt();
        add ad=new add();
        int result=ad.add(num1,num2);
        System.out.println("Sum of "+num1+" and "+num2+" is "+result);
    }
}
```

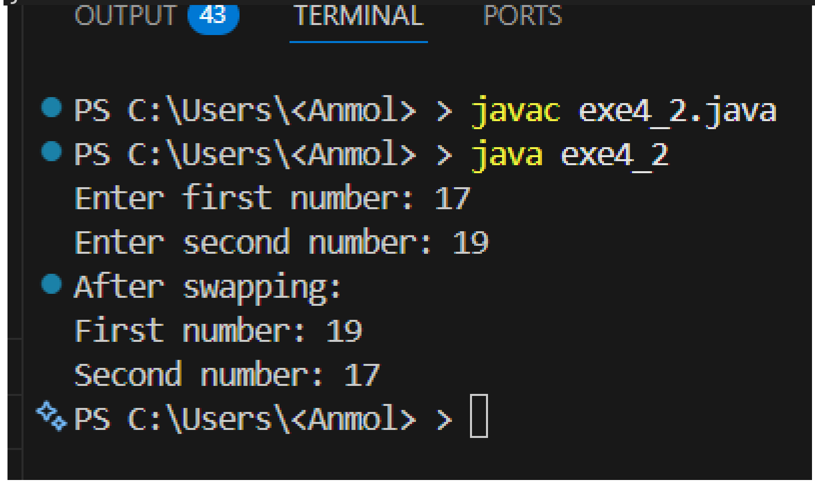


```
PS C:\Users\<Anmol> > javac exe4_1.java
PS C:\Users\<Anmol> > java exe4_1
Enter first number: 59
Enter second number: 78
Sum of 59 and 78 is 137
PS C:\Users\<Anmol> >
```

Observation: The program uses bitwise operators to perform addition without using arithmetic operators. The carry is calculated using the AND operator, and the sum is calculated using the XOR operator. The process continues until there are no more carries left to add.

Exe4_2

```java
//Write a program to swap two numbers without using a third variable.
import java.util.Scanner;
class swap
{
    void swap(int a,int b)
    {
        a=a+b;
        b=a-b;
        a=a-b;
        System.out.println("After swapping:");
        System.out.println("First number: "+a);
        System.out.println("Second number: "+b);
    }
}
class exe4_2
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter first number: ");
        int num1=sc.nextInt();
        System.out.print("Enter second number: ");
        int num2=sc.nextInt();
        swap sw=new swap();
        sw.swap(num1,num2);
    }
}
```

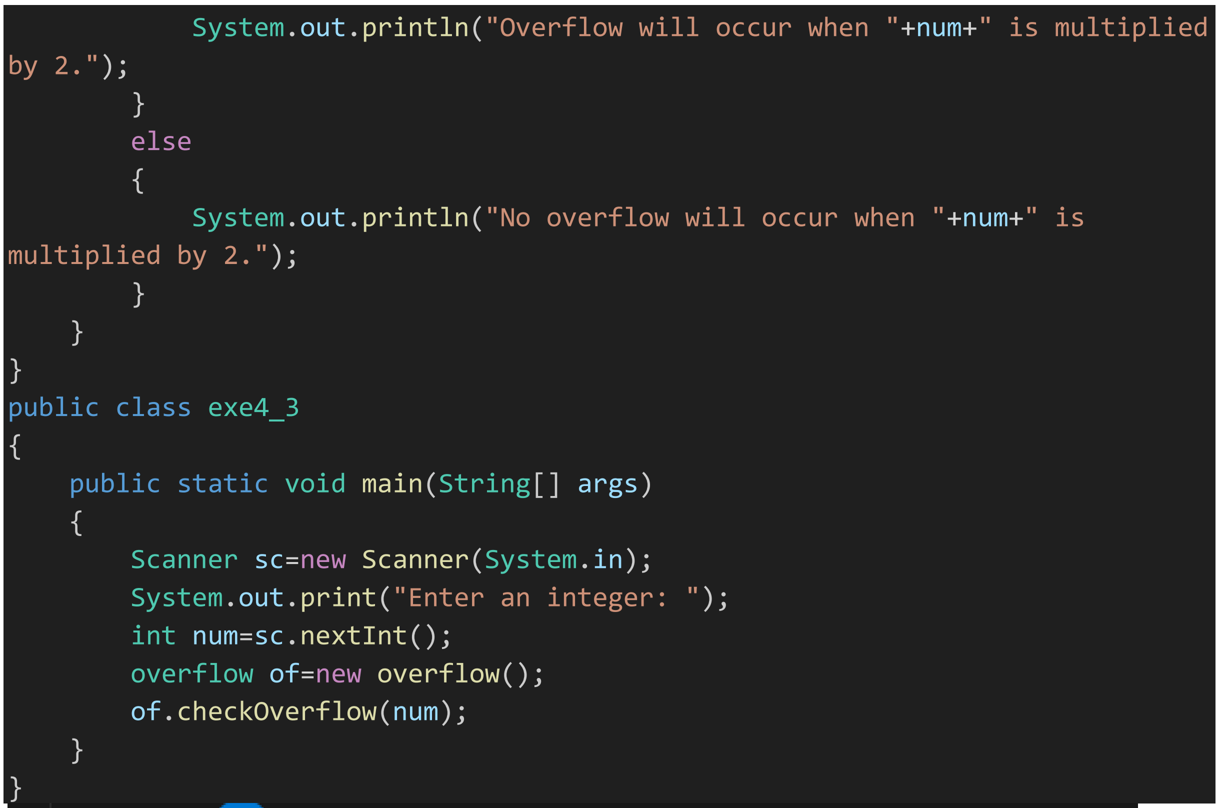


```
PS C:\Users\<Anmol> > javac exe4_2.java
PS C:\Users\<Anmol> > java exe4_2
Enter first number: 17
Enter second number: 19
After swapping:
First number: 19
Second number: 17
PS C:\Users\<Anmol> >
```

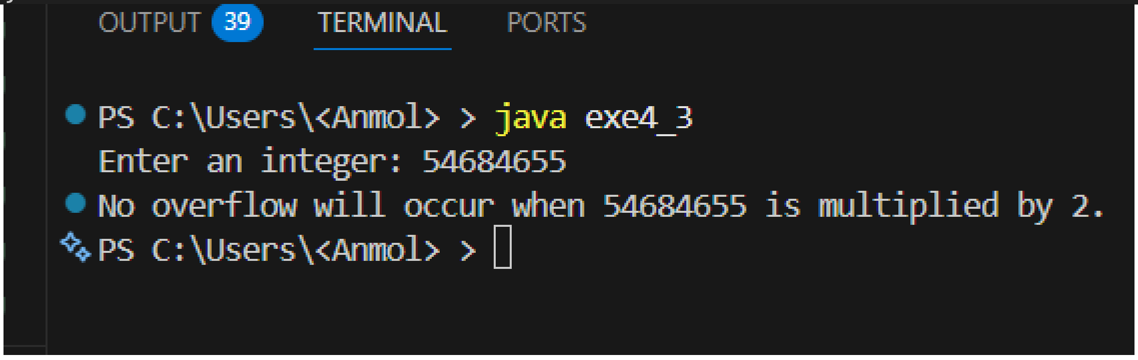
Observation: The program uses arithmetic operations to swap two numbers without using a third variable. The first step adds the two numbers and stores the result in 'a'. The second step subtracts 'b' from 'a' to get the original value of 'a' and stores it in 'b'. The third step subtracts the new value of 'b' from 'a' to get the original value of 'b' and stores it in 'a'. The final step prints the swapped numbers.

Exe4_3

```
//Write a program to check whether a given integer will overflow when
multiplied by 2 (use data type limits).
import java.util.Scanner;
class overflow
{
    void checkOverflow(int num)
    {
        if(num>Integer.MAX_VALUE/2||num<Integer.MIN_VALUE/2)
        {
```



```
            System.out.println("Overflow will occur when "+num+" is multiplied
by 2.");
        }
        else
        {
            System.out.println("No overflow will occur when "+num+" is
multiplied by 2.");
        }
    }
}
public class exe4_3
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int num=sc.nextInt();
        overflow of=new overflow();
        of.checkOverflow(num);
    }
}
```

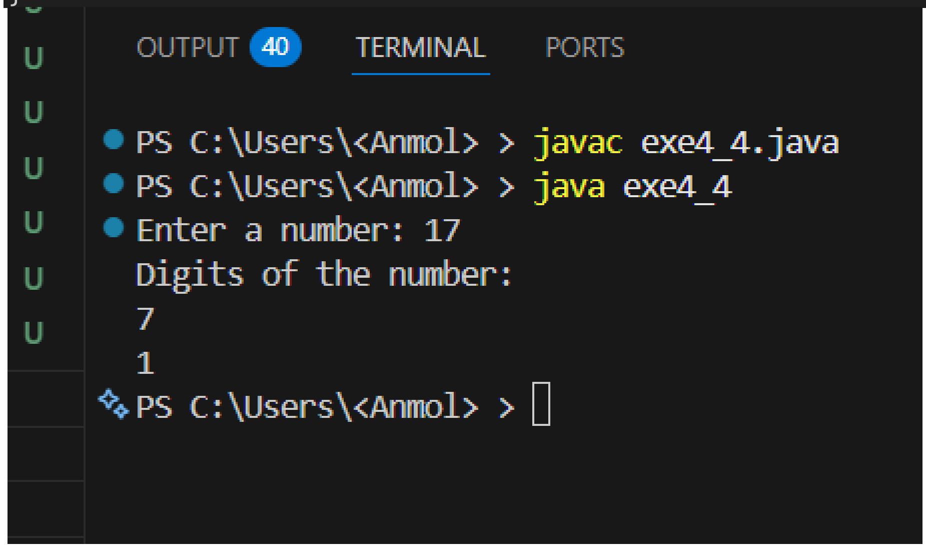


```
PS C:\Users\<Anmol> > java exe4_3
Enter an integer: 54684655
No overflow will occur when 54684655 is multiplied by 2.
PS C:\Users\<Anmol> >
```

Observation: The program checks whether the given integer will overflow when multiplied by 2 by comparing it to the maximum and minimum limits of the integer data type. If the number is greater than half of the maximum value or less than half of the minimum value, it indicates that multiplying by 2 will cause an overflow. Otherwise, it indicates that no overflow will occur.

Exe4_4

```java
//Write a program to extract and print each digit of a number using only arithmetic operators.
import java.util.Scanner;
class extract
{
    void extract(int num)
    {
        System.out.println("Digits of the number:");
        while(num>0)
        {
            int d=num%10;
            System.out.println(d);
            num=num/10;
        }
    }
}
public class exe4_4
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num=sc.nextInt();
        extract ex=new extract();
        ex.extract(num);
    }
}
```

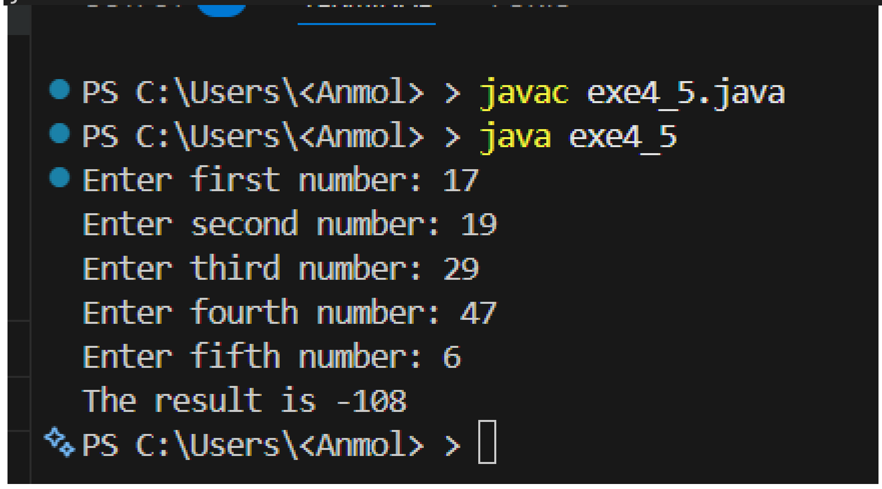


```
PS C:\Users\<Anmol> > javac exe4_4.java
PS C:\Users\<Anmol> > java exe4_4
Enter a number: 17
Digits of the number:
7
1
PS C:\Users\<Anmol> >
```

Observation: The program uses arithmetic operators to extract and print each digit of a number. The modulus operator (%) is used to get the last digit of the number, and the division operator (/) is used to remove the last digit from the number. This process continues until all digits have been extracted and printed.

exe4_5

```java
//Write a program to evaluate the expression a+b*(c-d)/e and demonstrate the
operator precedence by printing intermediate results.
import java.util.Scanner;
class evaluate
{
    void evaluate(int a,int b,int c,int d,int e)
    {
        int result=(a+b)*(c-d)/e;
        System.out.println("The result is "+result);
    }
}
public class exe4_5
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter first number: ");
        int a=sc.nextInt();
        System.out.print("Enter second number: ");
        int b=sc.nextInt();
        System.out.print("Enter third number: ");
        int c=sc.nextInt();
        System.out.print("Enter fourth number: ");
        int d=sc.nextInt();
        System.out.print("Enter fifth number: ");
        int e=sc.nextInt();
        evaluate ev=new evaluate();
        ev.evaluate(a,b,c,d,e);
    }
}
```

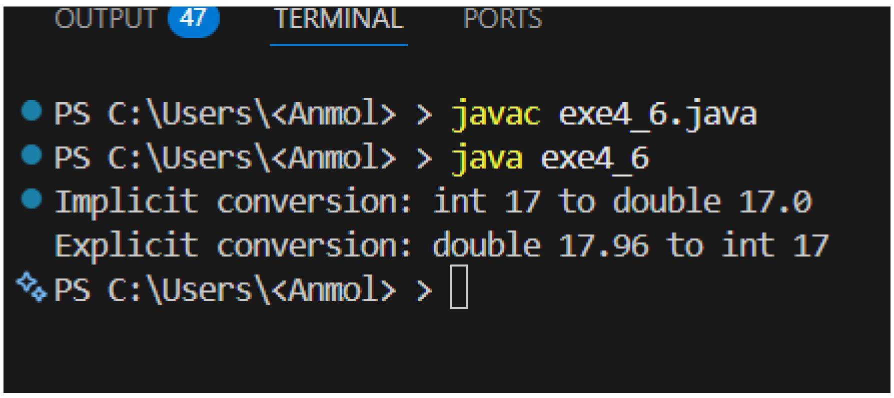


```
PS C:\Users\<Anmol> > javac exe4_5.java
PS C:\Users\<Anmol> > java exe4_5
Enter first number: 17
Enter second number: 19
Enter third number: 29
Enter fourth number: 47
Enter fifth number: 6
The result is -108
PS C:\Users\<Anmol> >
```

Observation: The program evaluates the expression $(a+b)*(c-d)/e$ and displays the result.

Exe4_6

```java
//Write a program to demonstrate implicit and explicit type conversion and show loss of precision.
class convert
{
    void implicit()
    {
        int num=17;
        double d=num;
        System.out.println("Implicit conversion: int " +num+ " to double "+d);
    }
    void explicit()
    {
        double d=17.96;
        int num=(int)d;
        System.out.println("Explicit conversion: double " +d+ " to int "+num);
    }
}
public class exe4_6
{
    public static void main(String[] args)
    {
        convert conv=new convert();
        conv.implicit();
        conv.explicit();
    }
}
```

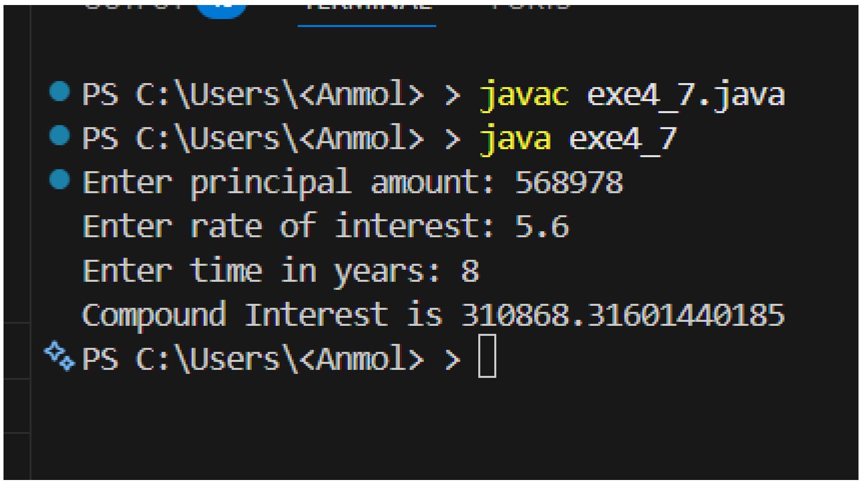


```
PS C:\Users\<Anmol> > javac exe4_6.java
PS C:\Users\<Anmol> > java exe4_6
Implicit conversion: int 17 to double 17.0
Explicit conversion: double 17.96 to int 17
PS C:\Users\<Anmol> >
```

Observation: The program demonstrates implicit and explicit type conversion. In the implicit conversion, an integer value is automatically converted to a double without any loss of precision. In the explicit conversion, a double value is explicitly cast to an integer, which results in a loss of precision as the decimal part is truncated.

Exe4_7

```
//Write a program to compute compound interest using appropriate data types
and arithmetic operators.
import java.util.Scanner;
class compound
{
    void compute(int p,double r,int t)
    {        double ci=p*Math.pow(1+(r/100.0),t)-p;
        System.out.println("Compound Interest is "+ci);
    }
}
public class exe4_7
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter principal amount: ");
        int principal=sc.nextInt();
        System.out.print("Enter rate of interest: ");
        double rate=sc.nextDouble();
        System.out.print("Enter time in years: ");
        int time=sc.nextInt();
        compound comp=new compound();
        comp.compute(principal,rate,time);
    }
}
```

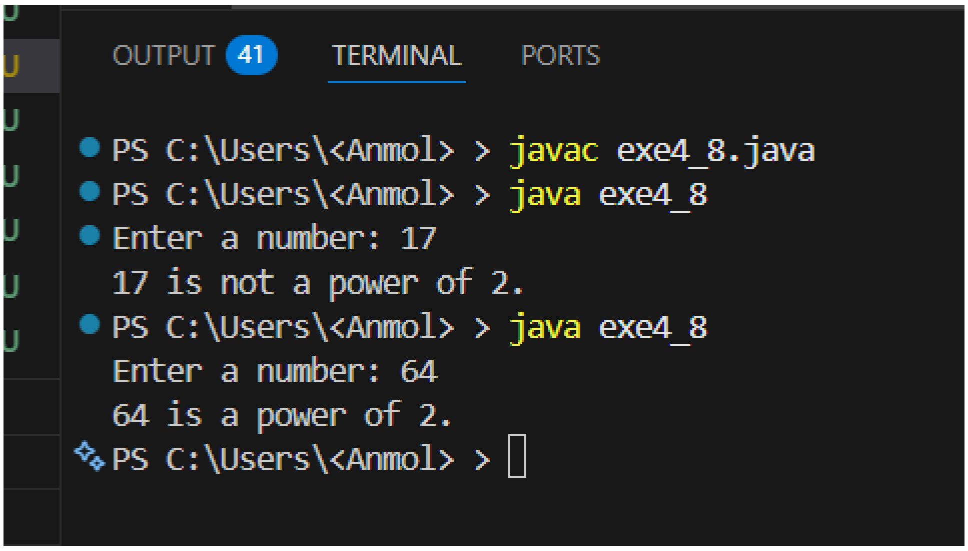


```
PS C:\Users\<Anmol> > javac exe4_7.java
PS C:\Users\<Anmol> > java exe4_7
Enter principal amount: 568978
Enter rate of interest: 5.6
Enter time in years: 8
Compound Interest is 310868.31601440185
PS C:\Users\<Anmol> >
```

Observation: The program calculates the compound interest based on the principal amount, rate of interest, and time period provided by the user. It uses the formula for compound interest and demonstrates the use of appropriate data types and arithmetic operators to compute the result.

Exe4_8

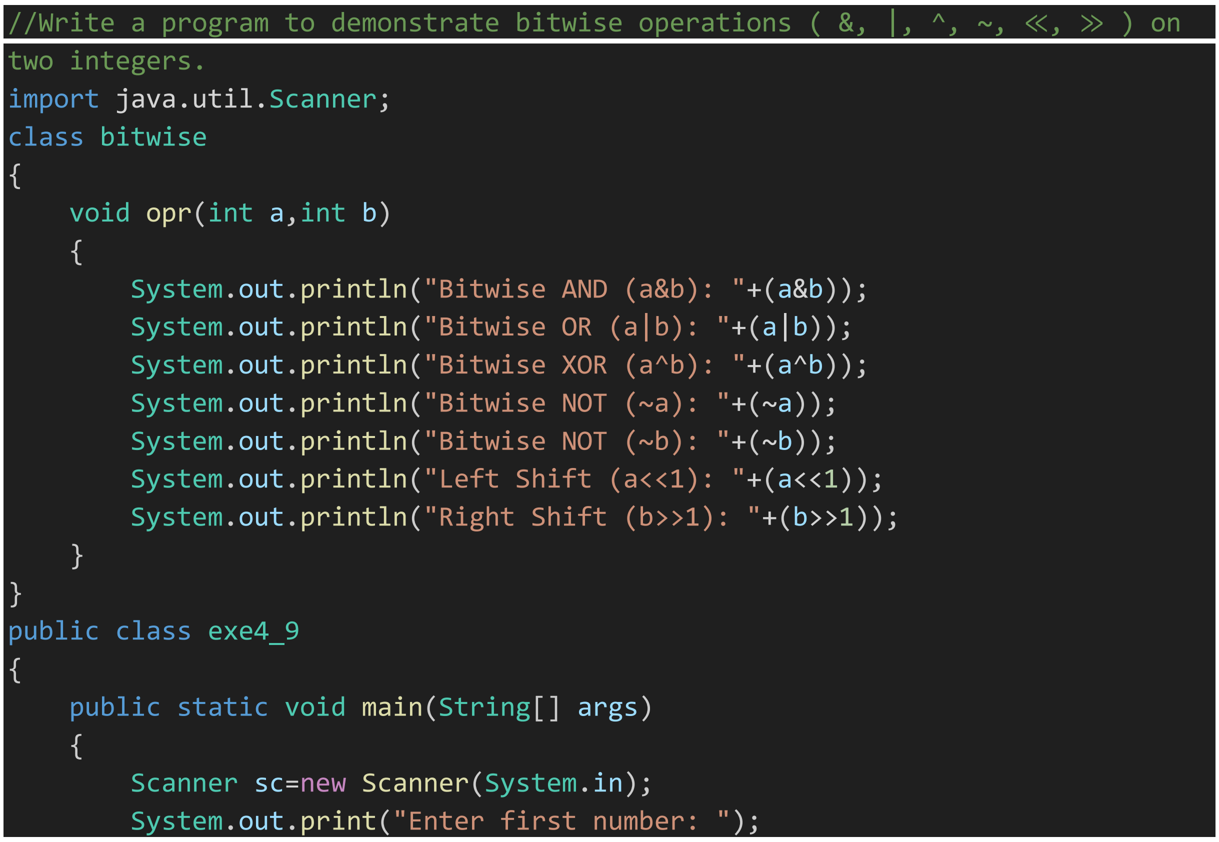
```
//Write a program to check whether a number is a power of 2 using bitwise
operators.
import java.util.Scanner;
public class exe4_8
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num=sc.nextInt();
        int r=num&(num-1);
        if(r==0)
        {
            System.out.println(num+" is a power of 2.");
        }
        else
        {
            System.out.println(num+" is not a power of 2.");
        }
    }
}
```



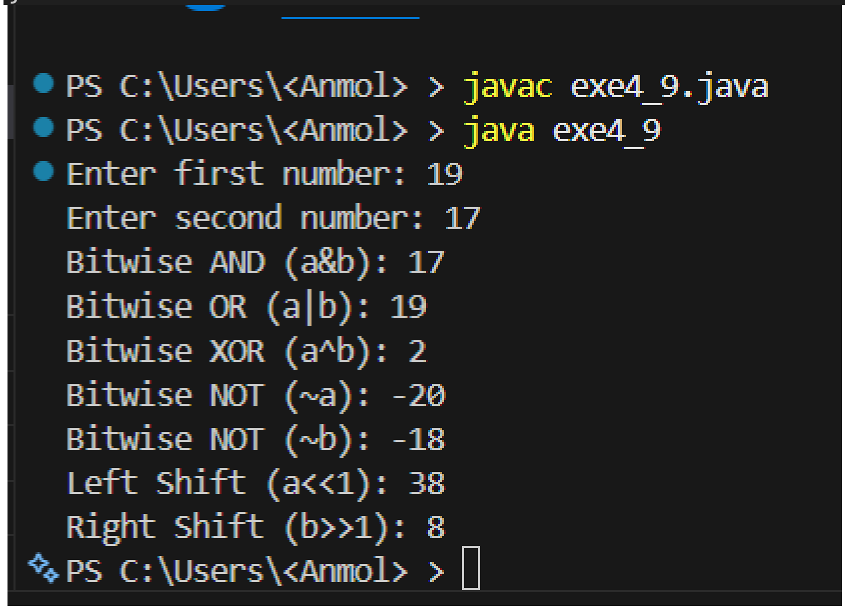
```
PS C:\Users\<Anmol> > javac exe4_8.java
PS C:\Users\<Anmol> > java exe4_8
Enter a number: 17
17 is not a power of 2.
PS C:\Users\<Anmol> > java exe4_8
Enter a number: 64
64 is a power of 2.
PS C:\Users\<Anmol> >
```

Observation: The program checks whether a number is a power of 2 by using bitwise operators. A number that is a power of 2 has only one bit set in its binary representation. By performing a bitwise AND operation between the number and one less than the number, we can determine if it is a power of 2. If the result is 0, it means that the number is a power of 2; otherwise, it is not.

Exe4_9



```
//Write a program to demonstrate bitwise operations ( &, |, ^, ~, ≪, ≫ ) on
two integers.
import java.util.Scanner;
class bitwise
{
    void opr(int a,int b)
    {
        System.out.println("Bitwise AND (a&b): "+(a&b));
        System.out.println("Bitwise OR (a|b): "+(a|b));
        System.out.println("Bitwise XOR (a^b): "+(a^b));
        System.out.println("Bitwise NOT (~a): "+(~a));
        System.out.println("Bitwise NOT (~b): "+(~b));
        System.out.println("Left Shift (a<<1): "+(a<<1));
        System.out.println("Right Shift (b>>1): "+(b>>1));
    }
}
public class exe4_9
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter first number: ");
        int num1=sc.nextInt();
        System.out.print("Enter second number: ");
        int num2=sc.nextInt();
        bitwise bw=new bitwise();
        bw.opr(num1,num2);
    }
}
```

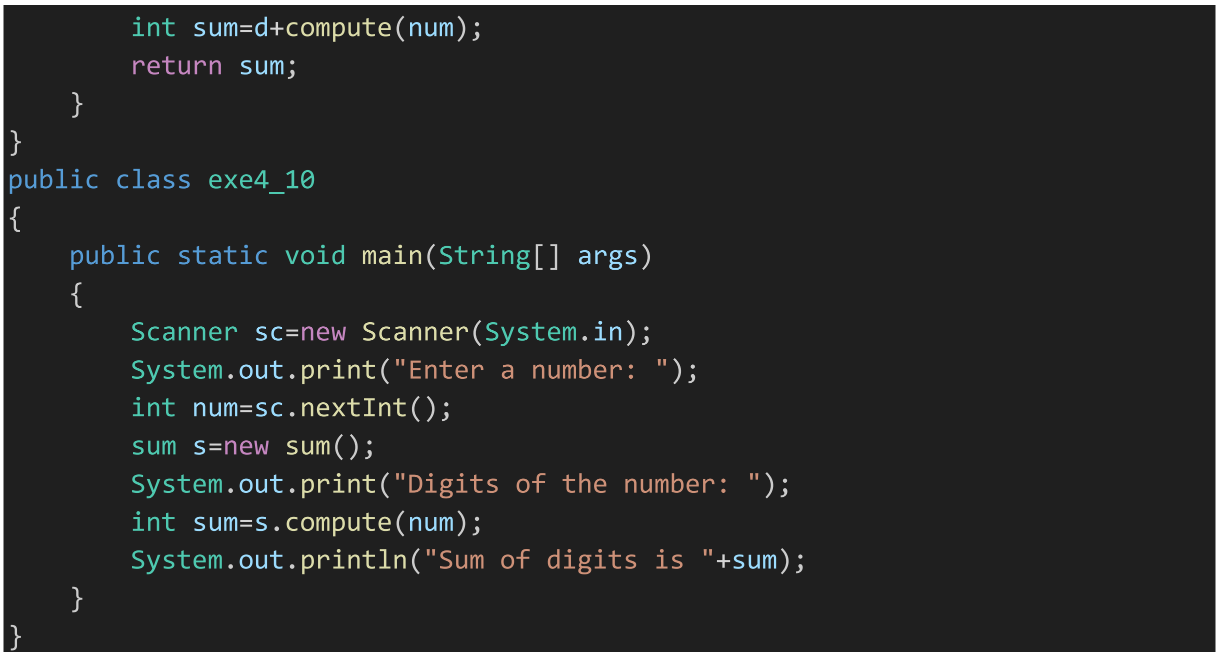


```
PS C:\Users\<Anmol> > javac exe4_9.java
PS C:\Users\<Anmol> > java exe4_9
Enter first number: 19
Enter second number: 17
Bitwise AND (a&b): 17
Bitwise OR (a|b): 19
Bitwise XOR (a^b): 2
Bitwise NOT (~a): -20
Bitwise NOT (~b): -18
Left Shift (a<<1): 38
Right Shift (b>>1): 8
PS C:\Users\<Anmol> >
```

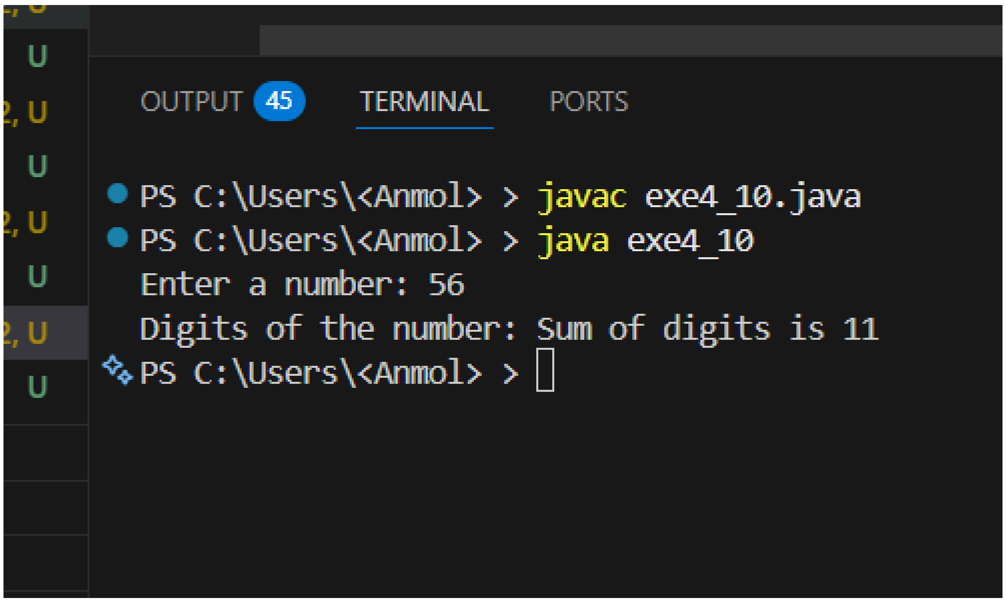
Observation: The program demonstrates bitwise operations on two integers. The output displays the results of various bitwise operations such as AND, OR, XOR, NOT, Left Shift, and Right Shift. Each operation is performed on the two input integers, and the results are printed to the console. This allows the user to understand how bitwise operators work in Java.

Exe4_10

```
//Write a program to compute the sum of digits of a number without using loops
(use arithmetic operators only).
import java.util.Scanner;
class sum
{
    int compute(int num)
    {
        if(num==0)
        {
            return 0;
        }
        int d=num%10;
        num=num/10;
```



```java
        int sum=d+compute(num);
        return sum;
    }
}
public class exe4_10
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num=sc.nextInt();
        sum s=new sum();
        System.out.print("Digits of the number: ");
        int sum=s.compute(num);
        System.out.println("Sum of digits is "+sum);
    }
}
```



```
PS C:\Users\<Anmol> > javac exe4_10.java
PS C:\Users\<Anmol> > java exe4_10
Enter a number: 56
Digits of the number: Sum of digits is 11
PS C:\Users\<Anmol> >
```

Observation: The program computes the sum of digits of a number without using loops by using recursion. The modulus operator (%) is used to get the last digit of the number, and the division operator (/) is used to remove the last digit from the number. The recursive function continues until the number becomes 0, at which point it prints the sum of the digits.