Course Name:	**Object Oriented Programming Methodology**	**Semester:**	**III**
Date of Performance:	10 /**08**/ 2023	**Batch No:**	B-2
Faculty Name:	**Prof.**	**Roll No:**	16010122151
Faculty Sign & Date:		**Grade/Marks:**	___/**25**

Experiment No: 1
Title: Perfect Number

Theory:

(Mention about data type, operator and flow statement)

Java Scanner Class

Java **Scanner class** allows the user to take input from the console. It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

Syntax

```
Scanner sc=new Scanner(System.in);
```

The above statement creates a constructor of the Scanner class having **System.inM** as an argument. It means it is going to read from the standard input stream of the program.
The **java.util** package should be import while using Scanner class.

Aim and Objective of the Experiment:

Define a class Perfect which accepts the range of numbers from the user. Create a static function check_per , which checks if the number is a perfect number or not and sends the result back to the main function which counts and displays the perfect numbers within that range.
Variations :
Implementation of Program with One class
Accessibility with static and non-static methods within class and outside class.

COs to be achieved:

CO2: Explore arrays, vectors, classes and objects in C++ and Java.

Tools used:

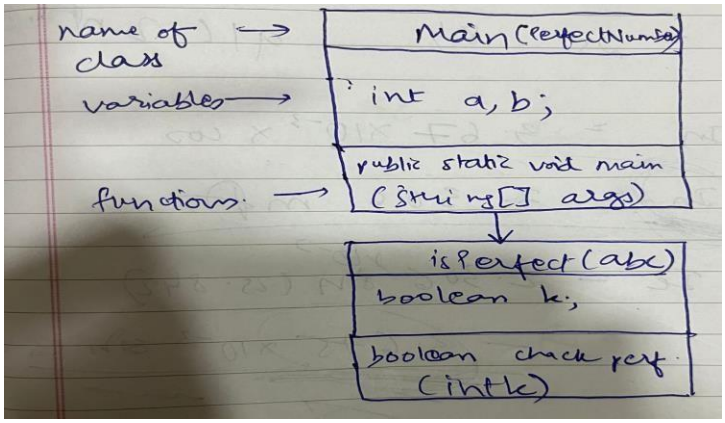
JDK, VScode / Eclipse

It also converts the Bytes (from the input stream) into characters using the platform's default charset.

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

3.
```
Scanner sc= new Scanner(System.in);    //System.in is a standard input stream
System.out.print("Enter first number- ");
int a= sc.nextInt();
```

Class Diagram:



Algorithm:

1. . Import Required Package:
   - Import the necessary package for input handling (`java.util`).

2. Define Class:
   - Create a class named `PerfectNumber`.

3. Define `main` Method:
   - Declare a `main` method to start the program.

4. Input Numbers:
   - Create a way to input two numbers, `a` and `b`, which define the range to check for perfect numbers.

5. Loop Through Range:
   - Start a loop that iterates through numbers from `a` to `b`.

6. Check for Perfection:
   - For each number within the loop:
     - Calculate the sum of its divisors (excluding itself).
     - Compare the sum with the current number.

7. Print Perfect Numbers:
   - If the sum of divisors is equal to the current number, print that the number is a perfect number.

8. Define Method (`isPerfect`):
   - Create a helper function called `isPerfect` that takes an integer as input and returns a boolean value.
   - Inside `isPerfect`:
     - Initialize a variable `sum` to store the sum of divisors.
     - Loop from `1` to half of the input number (since the largest possible divisor is half of the number):
       - Check if the number is divisible evenly by the current loop variable.
       - If so, add the loop variable to the `sum`.
     - Compare `sum` with the original input number:
       - If they are equal, return `true` (indicating a perfect number).
       - Otherwise, return `false`.

9. Loop Logic Continues:
   - The loop in step 5 continues until all numbers in the specified range have been checked.

10. End of Program:
    - The program execution completes when the loop finishes and all relevant information has been

printed.

This algorithm defines the logic for checking and identifying perfect numbers within a given range. It iterates through the range, calculates the sum of divisors for each number, and compares the sum with the number itself to determine if it's a perfect number. The `isPerfect` helper function encapsulates this divisibility check.

Code:

Using static function in same class

```
import java.util.*;
class perfectnumber
{
public static void main(String args[])
{
int a,b;
Scanner s=new Scanner(System.in);
System.out.println("enter number");
a=s.nextInt();
b=s.nextInt();
for (int i=a;i<=b;i++)
{
    if(abc(i)==true){
        System.out.println(i+" is a perfect number ");
    }

}}
static boolean abc(int k)
{
int i=1;
int sum=0;
while(i<=k/2)
{
if(k%i==0)
{
sum=sum+i;
}
i++;
}
if (sum==k)
{
 return true;
}
else
{
    return false;
```

```
}
}
}
```

Using static function in separate class

```
import java.util.*;
class perfectnumber
{
public static void main(String args[])
{
int a,b;
Scanner s=new Scanner(System.in);
System.out.println("enter range");
a=s.nextInt();
b=s.nextInt();
for (int i=a;i<=b;i++)
{
    if(perf.abc(i)==true){
        System.out.println(i+" is a perfect number ");
    }

}}
}
class perf
{
static boolean abc(int k)
{
int i=1;
int sum=0;
while(i<=k/2)
{
if(k%i==0)
{
sum=sum+i;
}
i++;
}
if (sum==k)
{
 return true;
}
else
```

```
{
    return false;
}
}
}
```

By creating object of separate class

```
import java.util.*;
class perfectnumber
{
public static void main(String args[])
{
int a,b;
perf x=new perf();
Scanner s=new Scanner(System.in);
System.out.println("enter range");
a=s.nextInt();
b=s.nextInt();
for (int i=a;i<=b;i++)
{
    if(x.abc(i)==true){
        System.out.println(i+" is a perfect number ");
    }

}}
}
class perf
{
boolean abc(int k)
{
int i=1;
int sum=0;
while(i<=k/2)
{
if(k%i==0)
{
sum=sum+i;
}
i++;
}
if (sum==k)
{
 return true;
}
```

```
else
{
    return false;
}
}
}
```

Post Lab Subjective/Objective type Questions:

1. Write a program to find the area and circumference of a circle using two classes.

```
import java.util.*;
class abc
{
public static void main(String args[])
{
float r;
Scanner s=new Scanner(System.in);
System.out.println("enter radius");
r=s.nextFloat();
System.out.println(circle.circum(r)+" is the circumference");
System.out.println(circle.area(r)+" is the area");

}}
class circle
{
static double circum(float k)
{
    double a=(4.28*k);
    return a;
}

static double area(float k)
{
    double b=(3.14*k*k);
    return b;
}
}
```

```
C:\Users\phobo\Desktop\java>java perfect
enter radius
5
21.400000000000002 is the circumference
78.5 is the area
```

2. Write the output of following program

```
public class BreakExample2 {
public static void main(String[] args) {
            //outer loop
            for(int i=1;i<=3;i++){
                    //inner loop
                    for(int j=1;j<=3;j++){
if(i==2&&j==2){
                            //using break statement inside the
inner loop
                            break;
                        }
                        System.out.println(i+" "+j);
                    }
            }
}
```

Output:

1 1
1 2
1 3
2 1
3 1
3 2
3 3

3. Why is java known as a platform independent language?

Java is known as a platform-independent language because it was designed with the principle of "write once, run anywhere" (WORA). This means that Java code can be written on one platform and executed on any other platform without modification. This platform independence is achieved through several key features of the Java programming language:

1. Java Virtual Machine (JVM): Java programs are compiled into an intermediate form called bytecode, which is platform-independent. When a Java program is executed, the JVM interprets and executes the bytecode. Each platform (Windows, macOS, Linux, etc.) has its own implementation of the JVM, allowing Java programs to run on any platform that has a compatible JVM.

2. Bytecode: Java source code is compiled into bytecode instead of native machine code. Bytecode is a low-level representation of the Java code that can be executed by the JVM. Since bytecode is platform-independent, it can be run on any system with a suitable JVM.

3. Standard Libraries: Java provides a comprehensive set of standard libraries that abstract the underlying operating system and hardware. These libraries offer a consistent set of functionalities across different platforms, making it easier for developers to write code that works across various environments.

4. No Pointers: Java eliminates the use of pointers, which are a source of platform-specific issues and potential security vulnerabilities. Instead, Java uses references, which are managed by the JVM and ensure more robust memory management.

5. Strict Language Specifications: Java has a strict and well-defined specification, which ensures consistent behavior across different implementations of the language and helps avoid platform-specific variations.

6. Security Model: Java's security model adds an additional layer of protection by controlling access to system resources. It enables Java programs to run in a sandboxed environment, mitigating potential security risks.

Conclusion: Hence we have implemented a program with static and non-static methods within and outside a class

Output:
enter range 0 100 6 is a perfect number 28 is a perfect number

Signature of faculty in-charge with Date: