Explanation for Modularity in programming:

Modularity in programming is a design principle that involves breaking down a program into separate, interchangeable modules. Each module encapsulates a specific piece of functionality and exposes a well-defined interface for other modules to use. Here’s how modularity aids in code reuse and organization:

* **Code Reuse**: Modules are designed to be self-contained, promoting reuse. Developers can take a module from one project and plug it into another, reducing the need to write new code for common tasks.
* **Organization**: Modularity helps in organizing code logically. By separating concerns, developers can focus on one aspect of the program at a time, making the codebase easier to understand and maintain.
* **Collaboration**: When working in teams, modularity allows multiple developers to work on different modules simultaneously without causing conflicts, enhancing productivity.
* **Debugging and Maintenance**: With modular code, identifying and fixing bugs becomes easier because the problem is usually contained within a single module. Similarly, updates or enhancements can be made to individual modules without affecting the rest of the application.

Therefore, modularity leads to cleaner, more manageable, and scalable codes.

Code:

package com.wipro.day03;

import java.util.Scanner;

public class Assign3 {

public static void main(String… pra) {

Scanner tin=new Scanner(System.***in***);

System.***out***.println("Enter a number:");

int number =tin.nextInt();

System.***out***.println("Factorial of " + number + " is: " + *factorial*(number));

System.***out***.println("The " + number + "th Fibonacci number is: " + *fibonacci*(number));

public static long factorial(int n) {

if (n < 0) {

System.***out***.println("Number cannot be negative");

long result = 1;

for (int i = 1; i <=n; i++) {

result \*= i;

return result;

public static int fibonacci(int number) {

if (number <= 1) {

return number;

int fib = 1;

int prevFib = 1;

for (int i = 2; i < number; i++) {

int temp = fib;

fib += prevFib;

prevFib = temp;

return fib;

}

}