***CASE STUDY NO. 2***

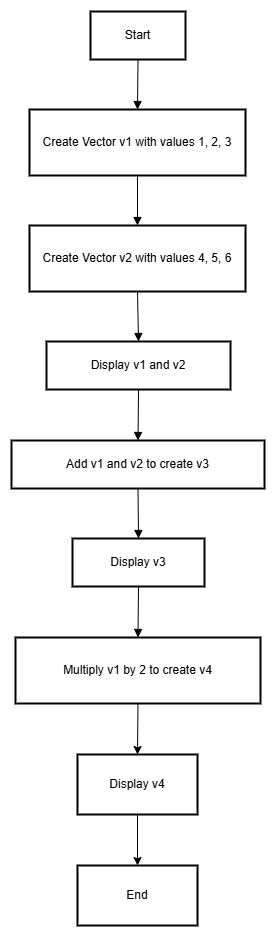
* **Title**
* Represent a vector using class and include appropriate methods to perform various tasks.
* **Theory Concepts Of Java/OOP Used in This Case Study**

1. **Class**: A class is a blueprint for creating objects and defines their properties and behaviors. It encapsulates data and methods. For instance, the Vector class in this example defines attributes and operations like addition and scalar multiplication for vectors.
2. **Object**: An object is a specific instance of a class that holds unique data. In the example, v1, v2, v3, and v4 are objects of the Vector class, each representing a unique vector with specific values.
3. **Encapsulation**: Encapsulation is the principle of bundling data and methods that operate on the data within a class while restricting direct access. In the Vector class, attributes x, y, and z are private, and controlled access is provided through methods like add and scalarMultiply.
4. **Constructor**: A constructor is a special method in a class that initializes the state of an object during its creation. In the Vector class, the constructor is used to set initial values for the x, y, and z attributes when a vector object is created.
5. **Method:** A method defines the behavior or actions of a class. For example, the add method in the Vector class performs vector addition, and the scalarMultiply method computes the result of multiplying the vector by a scalar.

* **Algorithm:**

1. **Start**
2. **Define the Vector class**:
   * Declare private integer fields x, y, and z.
   * Define a constructor to initialize the fields with given values.
   * Implement the add method to compute the sum of two vectors.
   * Implement the scalarMultiply method to multiply the vector by a scalar.
   * Implement the display method to print the vector components.
3. **Define the VectorOperations class**:
   * In the main method:  
     a. Create two Vector objects (v1 and v2) with specific values.  
     b. Display the components of v1 and v2.  
     c. Call the add method on v1 with v2 as the argument and store the result in v3.  
     d. Display the components of v3.  
     e. Call the scalarMultiply method on v1 with a scalar value and store the result in v4.  
     f. Display the components of v4.
4. **End**

* **Flowchart:**



* **Program/Code:**
* class Vector

{

    private int x, y, z;

    public Vector(int x, int y, int z)

    {

        this.x = x;

        this.y = y;

        this.z = z;

    }

    public Vector add(Vector v)

    {

        return new Vector(this.x + v.x, this.y + v.y, this.z + v.z);

    }

    public Vector scalarMultiply(int scalar)

    {

        return new Vector(this.x \* scalar, this.y \* scalar, this.z \* scalar);

    }

    public void display()

    {

        System.out.println("Vector: (" + x + ", " + y + ", " + z + ")");

    }

}

public class VectorOperations

{

    public static void main(String[] args)

    {

        Vector v1 = new Vector(1, 2, 3);

        Vector v2 = new Vector(4, 5, 6);

        System.out.println("=== Original Vectors ===");

        v1.display();

        v2.display();

        Vector v3 = v1.add(v2);

        System.out.println("\n=== After Addition ===");

        v3.display();

        Vector v4 = v1.scalarMultiply(2);

        System.out.println("\n=== After Scalar Multiplication ===");

        v4.display();

    }

}

* **Output:**
* === Original Vectors ===

Vector: (1, 2, 3)

Vector: (4, 5, 6)

=== After Addition ===

Vector: (5, 7, 9)

=== After Scalar Multiplication ===

Vector: (2, 4, 6)

* **Conclusion**
* This case study illustrates how a vector can be modeled using a class in Java. The Vector class encapsulates properties like components (x, y, and z) and provides methods to perform operations such as addition and scalar multiplication. By leveraging OOP principles like encapsulation and methods, this approach simplifies complex mathematical operations into reusable and readable code. The structure is both efficient and expandable for additional functionality.