Coupling and Cohesion

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Types of Coupling:

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don’t contain tramp data. Example-customer billing system.

Stamp Coupling In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.

- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Coupling Example

```
class Volume
{
    public static void main(String args[])
    {
        Box b = new Box(5,5,5);
        System.out.println(b.volume);
    }
}
class Box
{
    public int volume;
    Box(int length, int width, int height)
    {
        this.volume = length * width * height;
    }
}
```

Type of possible coupling in the above code with justification:

The above code exhibits data coupling.

Data coupling refers to the situation where two modules share data through parameters, global variables or return values. In the given code, the Volume class creates an instance of the Box class (Box b = new Box(5,5,5)) and accesses its volume variable directly (System.out.println(b.volume)).

This creates a direct dependency between the Volume and Box classes, where the Volume class is dependent on the volume variable of the Box class. This is an example of data coupling, as the Volume class is sharing data with the Box class through the volume variable.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

- **Types of Cohesion:**
- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional

- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.

- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

Example of Cohesion

Example: Suppose we have a class that multiplies two numbers, but the same class creates a pop-up window displaying the result. This is an example of a low cohesive class because the window and the multiplication operation don't have much in common. To make it high cohesive, we would have to create a class Display and a class Multiply. The Display will call Multiply's method to get the result and display it. This way to develop a high cohesive solution.

```
class Multiply {

    int a = 5;
    int b = 5;

    public int mul(int a, int b)
    {
        this.a = a;
        this.b = b;
        return a * b;
    }
}

class Display {
    public static void main(String[]
args)
    {
        Multiply m = new Multiply();
        System.out.println(m.mul(5,
5));
    }
}
```

Solution

The type of cohesion in this code is functional cohesion.

Functional cohesion occurs when the methods or functions within a module are related by a single task or functionality. In this code, the mul method of the Multiply class performs a single task, which is multiplying two integers and returning the result. The variables a and b are also related to this task, as they are the operands being multiplied.

Therefore, the Multiply class exhibits functional cohesion.