4. **(a)** **Modular programming** is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

or

In [software design](https://en.wikipedia.org/wiki/Software_design), **modularity** refers to a logical partitioning of the "software design" that allows complex software to be manageable for the purpose of implementation and maintenance. The logic of partitioning may be based on related functions, implementation considerations, data links, or other criteria.

**Encapsulation** is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of bundling data and methods that work on that data within one unit. Java supports it v well through classes.

**(b)** Inheritance dramatically improves adaptability. Components may be adapted without change by deriving a sub-class and modifying that derived class. Inheriting attributes from super-classes weakens cohesion

**Cohesion**

A measure of how well a component or module 'fits together'

A component should implement a single logical entity or function

Cohesion is a desirable design component attribute as when a change has to be made, it is localised in a single cohesive component

Various levels of cohesion have been identified

**Coupling**

A measure of the strength of the inter-connections between system components

Loose coupling means component changes are unlikely to affect other components

Shared variables or control information exchange lead to tight coupling

Loose coupling can be achieved by state decentralisation (as in objects) and component communication via parameters or message passing

(c) **Interfaces**

An interface in UML is a named set of operations shown as a stereotyped class

Generalization can be defined between interfaces

Realizing an Interface

A class realizes an interface if it provides implementations of all the operations in Java we say it implements an interface UML provides two equivalent ways of showing this relationship

Interface Dependency

A class can be dependent on an interface. This means that it only makes use of the

operations defined in that interface