Principles of Object-Oriented Systems

The conceptual framework of object-oriented systems is based upon the object model. There are two categories of elements in an object-oriented system:

**Major Elements**: By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

**Minor Elements**: By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are:

- Typing
- Concurrency
- Persistence

**Abstraction**

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Grady Booch has defined abstraction as follows:

“An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.”

**Example**: When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

**Encapsulation**

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

**Modularity**

Modularity is the process of decomposing a problem program into a set of modules so as to reduce the overall complexity of the problem. Booch has defined modularity as:

“Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.”

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter-module interaction or coupling is low.

**Hierarchy**
In Grady Booch’s words, “Hierarchy is the ranking or ordering of abstraction”. Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. It uses the principle of “divide and conquer”. Hierarchy allows code reusability.

The two types of hierarchies in OOA are:

- **“IS-A” hierarchy**: It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose “is-a” flower.

- **“PART-OF” hierarchy**: It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a “part-of” flower.

**Typing**

According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are:

- **Strong Typing**: Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.

- **Weak Typing**: Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

**Concurrency**

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

**Persistence**

An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.