# OOS – Objektno Orijentirani Sistem

Znamo da Objektno-Orjentirano Modeliranje (OOM) tehnika vizualizira stvari u aplikaciji koristeći modele organizirane oko objekata. Bilo koji softverski razvoj kreće se kroz sljedeće korake –

* Analiza,
* Dizajn
* Implementacija.

U objektno-orjentiranom softverskom inžinjerstvu, softverski razvijatelj identificira i organizira aplikaciju u smislu objektno-orjentiranih koncepta s obzirom na njihovu završnu izvedbu u bilo kojem proogramskom jeziku ili softverskom alatu.

Faze u Objektno-Orijentiranom Razvoju Softvera

Glavne faze u razvoju softvera koristeći objektno-orjentirane metode su objektno-orjentirana analiza, objektno-orjentirani dizajn, objektno-orjentirana implementacija.

Objektno-Orjentirana analiza

U ovoj fazi, problem je formuliran, korisnički zahtjevi su određeni, i objekt je izgrađen nalik na objekte u pravome svjetu. Analiza producira model ovisno o tome kako bi željeni sistem trebao funkcionirati i kako bi trebao biti razvijen. Modeli ne ukljucuju bilo kakve detalje u implementaciji tako da mogu biti jasni nestručnim osobama.

Objektno-Orjentirani dizajn

Objektno-orjentirani dizajn uključuje dvije glavne faze, dizajn sistema i objekta.

**Dizajn sistema**

U ovoj fazi dizajnira se kompletna arhitektura željenog sistema. Sustav je predočen kao set interaktivnih podsustava

**System Design**

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

**Object Design**

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether −

* new classes are to be created from scratch,
* any existing classes can be used in their original form, or
* new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

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.