**What is Low Level Design or LLD – Learn System Design**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 Feb, 2023

 Read

 Discuss

 Courses

 Practice

 Video

***LLD,****as the name suggests, stands for****low-level design****. It is a component-level design process that follows step by step refinement process. The input to LLD is HLD.*

LLD describes class diagrams with the help of methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document. It provides us with the structure and behavior of class as different entities have different character sets. From this design, it is easy for a developer to write down logic and henceforth the actual code for it.



*What is Low Level Design or LLD*

**Class Diagram in LLD:**In this diagram we basically list out all the Here we list out all the entities which can be part of components. Class diagrams are made as it becomes easier for developer to convert it into code.

For instance:

User Servcie

<-- User

<--Profile

<--ID

***Remember:****Low-level designing is also known as****object-level designing****or****micro-level****or****detailed designing****.*

[**How is LLD different from HLD**](https://www.geeksforgeeks.org/difference-between-high-level-design-and-low-level-design/)

As studied, [High Level Design or HLD](https://www.geeksforgeeks.org/what-is-high-level-design-learn-system-design) is a general system design where we do tradeoffs between different frameworks, components, and different databases and we choose the best of them. Here we define the components and how these components will be communicating with one another. Hence here we are bothered with generic stuff as follows and not bothered about the code.

1. Selection of components, platforms, and different tools.
2. Database design
3. Brief description of relationships between services and modules.

**How to form LLD from HLD?**

As studied above, input for framing low-level design (LLD) is HLD. Here in LLD, we take care of how our components will look like, the structure possessed by different entities, and how different entities will have their responsibility(operations supported). For this conversion, we use [Unified Modelling Language (UML) diagrams](https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/). Adding to these diagrams we use [OOPS principles](https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/) and [SOLID principles](https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/) while designing. Hence, using these 3 paradigm we can convert any HLD to LLD so as to get implemented.

**Roadmap to Low-level Designing**

In order to bridge concepts of LLD with real code let us In order to understand how to design any low-level diagram let us understand via the steps:

**1. Object-oriented Principles**

The user requirement is processed by using concepts of OOPS programming. Hence it is recommended to have a strong grip on OOPS concepts prior to moving ahead in designing any low-level system.

Object-oriented programming concept 4 pillars are must-have to go start learning low-level designing and the programmer should be very well versed with these 4 pillars namely as follows:

Inheritance, encapsulation, polymorphism, and abstraction. Within polymorphism, we should be clear-cut with compile-time ad run-time polymorphism. Programmers should be absolutely clear about the OOPS concepts to depth right to classes, and objects because OOPS is the foundation on which low-leveling on any system is based.

Acing low-level design is ‘extremely subjective’ because we have to optimally use these concepts while coding to build a low-level system via implementing coding software entities(classes, functions, modules, etc)

**2. Process of Analyzing and design**

It is analyzing phase which is our 1st step where we are claying real-world problems into object-world problems using OOPS concepts and SOLID principles.

**3. Design Patterns**

Now implementation of our above object world problem is carried out with help of design patterns. These are the set of practices indeed the solutions that are already proposed checked against test cases. We just need to map them and implement them asities as per the circumstance.

With the usage of objects, classes, SOLID principles, and UML diagrams, we can model complex problems to code while designing low-level systems. And at the same time, there exists a scenario that does not exist in the real world but has to be coded to get solved at low-level designing. For these certain sets of specified problems. There are solutions to these where these problems are already been solved in different complex oriented codes to the problem. Hence the definition is as follows:

*Each problem describes a problem that occurs over and over multiple times in the environment. And the core of the solution to the problem can be used a million times without ever doing it.*

**Why there is a need for design patterns?**

These problems have occurred over and over again corresponding to which these solutions have been laid out. These problems are been faced and solved by expert designers in the world of programming and the solutions are robust over time saving a lot of time and energy. Hence the complex and classic problems in the software world are being solved by tried and tested solutions. Here we do not look out for solutions that are the so-called design patterns.

*Tip: It is strongly recommanded to have good understanding of common design patterns to have hold over low-level designing.*

**Different Types of Design Patterns**

There are widely varied many types of design patterns while many

Let us discuss 4 types of design patterns that are extensively used globally

* Factory Design Pattern
* Abstract factory Pattern
* Singleton Pattern
* Observer Pattern

It is also recommended to study the below 5 design patterns as these below 5 patterns are less frequently used in the industry

**4. UML Diagram**

They are 2 types of UML Diagrams:

1. Structural UML diagram: These types of diagrams basically defines how different entities and objects will be structured and defining the relationship between them. They are helpful in representing  how components will appear with respect to structure.
2. Behavioral UML diagram: These types of diagrams basically defines what are the different operations that it       supports. Here different behavioral UML showcases different behavioral of

*Tip: Important UML diagrams used by developers frequently are as follows:*

* *Class diagram from Structural UML Diagram*
* *Sequence, Use case and Activity from Behavioral UML Diagram.*

**5s. SOLID Principles**

These are sets of 5 principles(rules) that are strictly followed as per requirements of the system or requirements for optimal designing.

In order to write scalable, flexible, maintainable, and reusable code:

1. Single-responsibility problem (SRP)
2. Open-closed principle (OCP)
3. Liskov’s Substitution Principle(LSP)
4. Interface Segregation Principle (ISP)
5. Dependency Inversion Principle (DIP)

We need to keep checking while coding and implementing our designing classes as per need if we need to make our classes extendable then we need to talk about the open-closed principle. Or if our requirement is to design a class with a single responsibility principle.