LLD Day - 04

② Builder Design Pattern: Desktops

- RAM
- Processor
- Monitor
- Keyboard
- Mouse
- Speaker
- Printer

Builder → Abstract class / Interface.

* build RAM()
* build Processor()
⋮

The Factory Pattern is used when the exact object type should be chosen at runtime. Instead of calling constructors directly, you can use a factory pattern that returns the right subclass based on some condition. It hides creation logic and centralizes object instantiation. But when objects need many optional fields or detailed configuration, factories become harder to manage

P.T.O]

The Builder Patterns solves this by letting you construct complex objects step-by-step. It uses clean, chainable methods to set only the fields you need and then produce the final object with build(). This avoids long constructors & keeps complex object creation clean and readible.

Factories decide which object to create;
Builders control how to create it with many parameters.

Prototype & Singleton Pattern:

① Prototype: This pattern creates new objects by cloning an existing object instead of building one from scratch. You keep a "prototype" object and duplicate it whenever required. It is useful when object creation is expensive (or) complex, and copying is faster than construction.

② Singleton : This Pattern ensures that only one instance of class exist in the entire application and provides a global access point to it. It is commonly used for shared resources like configuration managers, loggers, or database connections.

Analyse codebase & understand briefly using Vehical Creation System Example.