The downsides of composition are that it may hide the rigidity of elements and may be difficult for others to understand. With, for example, a 2D point class and wanting to extend it to higher dimensions, you should add (at least) z getter/setter, modify getDistance() and, possibly, add a getVolume() method. So you have 101 object elements: status and related behavior.

It assumes that a developer-minded configurative has defined the getDistance (x, y) -> double method and will now determine the getDistance method (x, y, z) -> double. Or thinking in general, they might select getDistance (lambdaGeneratingACoordinateForEveryAxis()) -double >. Then they'll probably write createTwoDimensionalPoint() and createThreeDimensionalPoint() (or perhaps createNDimensionalPoint(n)) that will compile the diverse state and behavior.

The developer using the OO mindset will use inheritance. The same amount of complexity in the implementation of scope properties, less complicated in terms of initialization of the object (the constructor takes care of it versus the manufacturer's method), but not flexible in terms of what can be configured.

Now think about it from an understandable/readable point of view. To understand the configuration, it contains a large number of functions that are programmatically formed within another function. Therefore, there is little in terms of "code" "fixed" code (files, keywords, etc.) that makes the relationship between Z and distance () jump. In the OO world, you have a big red light that flashes big telling you the hierarchy. In addition, you have a mainly universal vocabulary for discussing structure, widely known graphical symbols, natural hierarchy (at least for individual heredity), etc.

On the other hand, on the other hand, the famous and clearly constructed factory method will often illustrate more of the sometimes ambiguous relationships between the state and behavior, since the syntactic mindset facilitates the functional code (that is, the code that passes the state through the parameters, not through ****this**** ).

In a professional environment with experienced developers, the flexibility of the composition generally outperforms its more abstract nature. However, one should never underestimate the importance of inclusivity, especially in teams that have varying degrees of experience and/or high levels of rotation.

2

#include <string>#include <iostream>#include <thread>

using namespace std;

// The function we want to execute on the new thread.void task1(string msg){

cout << "task1 says: " << msg;

}

int main(){

// Constructs the new thread and runs it. Does not block execution.

thread t1(task1, "Hello");

// Do other things...

// Makes the main thread wait for the new thread to finish execution, therefore blocks its own execution.

t1.join();

}

3.

#### **1. Unit Testing**

Every code consists of units or blocks of code that perform collectively to achieve a singular task. This singular task might not be the end motive of the entire application but a step towards achieving the same. The testing frameworks in this category look largely at the aspect of associated control of the data, procedure of usage, and operations, though just not restricted to only these. In the C++ testing framework, this genre of testing gets the maximum examples as most of the application building lies on the unit testing.

* + ****API Sanity Checker****: The framework looks into generating reasonable input arguments directly from the library header files so that every API function is tested.
  + ****CppUnit****: Using this framework, it is easy for an application with C and C++ sources to be easily tested with the least source modification. This framework runs the tests in the suite and has some advanced filters that allow easy continuous reporting integration systems.
  + ****Google Test****: One of the most widely used frameworks that have the trust of Google. This framework executes tests cases on the xUnit architecture, and this as well allows developers to unit test with the least code modification. OpenCV is a computer vision library that uses Google Test for all its unit testing purpose.
  + ****UnitTest++****: This is the lightweight unit testing framework that is also widely used in C++. This framework focuses on keeping it simple, portable without compromising on speed. This framework applies to all 3 platforms, viz. Windows, Linux, and Mac OS X.

#### **2. Integration Testing**

Often abbreviated as I&T, the modules that form the units in the above type of testing are collectively combined to perform the testing as a group. This looks into the compliance of the system when all the modules are used together, along with the adherence to the specified functionalities of the application as a whole. This step happens after the unit testing is performed. Some of the frameworks in C++ for integration testing (although one should know that most of these do support unit testing as well but are majorly used for integration testing) are:

* + ****VectorCAST/C++****: This is one of the proprietary frameworks that allow combo the benefit of building test harnesses to perform unit testing and integration testing. This tool also allows execution trace, so that debugging is assisted.
  + ****Cantata++****: This is a commercial program written for dynamic testing and is being marketed by QA Systems. The Ide is based on Eclipse and allows developers to perform unit and integration tests under the same hood.

#### **3. Regression Testing**

In this type of testing, all the functional and non-functional tests are run from the previously developed modules that already existed as the feature. We are sure that the older features are still usable even if we expand our horizon of the feature implementation in the application. In this, mostly the unit test frameworks are executed to test out the earlier written test cases.

#### **4. Acceptance Testing**

This genre of testing relates to evaluating the compliance of the system to the business demands so that it can be evaluated if it needs to be accepted or rejected. This is to do with mostly running some tests to make sure that the requirements under the contract are met. UAT (User Acceptance Testing) is typically performed in this regard to make sure that the end-user verifies and accepts the feature(s). not much framework exists and is mostly written as per the need of the application.

#### **5. Performance Testing**

Now that the application runs and meets the expectations, it is also equally important for this application to run at optimized speed, have high responsiveness, have no interference on the stability, and the application is performing as expected under a full capable workload.

* + ****CPPOCL/test****: This framework allows developers to identify tests as functions, passing or failing the test on the basis of a threshold in performance. This also allows custom logging to make sure that the logs can be made interpretable!

4.

## Definition[[edit](https://en.wikipedia.org/w/index.php?title=Architectural_pattern&action=edit&section=1" \o "Edit section: Definition)]

Even though an architectural pattern conveys an image of a system, it is not an architecture. An architectural pattern is a concept that solves and delineates some essential cohesive elements of a software architecture. Countless different architectures may implement the same pattern and share the related characteristics. Patterns are often defined as "strictly described and commonly available".[[3]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-3)[[4]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-4)

## Architectural style[[edit](https://en.wikipedia.org/w/index.php?title=Architectural_pattern&action=edit&section=2" \o "Edit section: Architectural style)]

Following traditional building architecture, a 'software [architectural style](https://en.wikipedia.org/wiki/Architectural_style" \o "Architectural style)' is a specific method of construction, characterized by the features that make it notable.

An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined.[[5]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-SG-5)

An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.[[1]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-TMD-1)

Some treat architectural patterns and architectural styles as the same,[[6]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-MSDN-6) some treat styles as specializations of patterns. What they have in common is both patterns and styles are idioms for architects to use, they "provide a common language"[[6]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-MSDN-6) or "vocabulary"[[5]](https://en.wikipedia.org/wiki/Architectural_pattern" \l "cite_note-SG-5) with which to describe classes of systems.

The main difference is that a pattern can be seen as a solution to a problem, while a style is more general and does not require a problem to solve for its appearance.

## Examples[[edit](https://en.wikipedia.org/w/index.php?title=Architectural_pattern&action=edit&section=3" \o "Edit section: Examples)]

Here is a list of architecture patterns, and corresponding [software design patterns](https://en.wikipedia.org/wiki/Software_design_pattern" \o "Software design pattern) and [solution patterns](https://en.wikipedia.org/wiki/Solution_architecture" \o "Solution architecture).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub-domain area** | **Architecture pattern** | **Software design patterns** | **Solution patterns** | **Related patterns** |
| [Data integration](https://en.wikipedia.org/wiki/Data_integration" \o "Data integration)/[SOA](https://en.wikipedia.org/wiki/Service-oriented_architecture" \o "Service-oriented architecture) | * [ETL (data extraction transformation and loading)](https://en.wikipedia.org/wiki/Extract,_transform,_load" \o "Extract, transform, load) | * [Change data capture](https://en.wikipedia.org/wiki/Change_data_capture" \o "Change data capture) * Near real-time ETL * Batch ETL * [Data discovery](https://en.wikipedia.org/wiki/Data_discovery" \o "Data discovery) | * [Error handling](https://en.wikipedia.org/wiki/Error_handling" \o "Error handling) * [Job scheduling](https://en.wikipedia.org/wiki/Job_scheduling" \o "Job scheduling) * [Data validation](https://en.wikipedia.org/wiki/Data_validation" \o "Data validation) * [Slowly changing dimensions load](https://en.wikipedia.org/wiki/Slowly_changing_dimension" \o "Slowly changing dimension) | * [EAI](https://en.wikipedia.org/wiki/Enterprise_application_integration" \o "Enterprise application integration) * Master data hub * [Operational data store](https://en.wikipedia.org/wiki/Operational_data_store" \o "Operational data store) (ODS) * [Data mart](https://en.wikipedia.org/wiki/Data_mart" \o "Data mart) * [Data warehouse](https://en.wikipedia.org/wiki/Data_warehouse" \o "Data warehouse) |
| * [MFT](https://en.wikipedia.org/wiki/Managed_file_transfer" \o "Managed file transfer) |  |  |  |
| * [EAI](https://en.wikipedia.org/wiki/Enterprise_application_integration" \o "Enterprise application integration)/[ESB](https://en.wikipedia.org/wiki/Enterprise_service_bus" \o "Enterprise service bus) | * [Publish/subscribe](https://en.wikipedia.org/wiki/Publish/subscribe" \o "Publish/subscribe) * [Request/reply](https://en.wikipedia.org/wiki/Request-response" \o "Request-response) * [Message exchange patterns](https://en.wikipedia.org/wiki/Messaging_pattern" \o "Messaging pattern) | * One-way * Synchronous request/response * Basic callback * Claim check | * [SOA](https://en.wikipedia.org/wiki/Service-oriented_architecture" \o "Service-oriented architecture) |
| [Data architecture](https://en.wikipedia.org/wiki/Data_architecture" \o "Data architecture) | * [Transaction data stores](https://en.wikipedia.org/wiki/Online_transaction_processing" \o "Online transaction processing) (TDS/OLTP) * [Master data store](https://en.wikipedia.org/wiki/Master_data_management" \o "Master data management) * [Operational data store](https://en.wikipedia.org/wiki/Operational_data_store" \o "Operational data store) * [Data mart](https://en.wikipedia.org/wiki/Data_mart" \o "Data mart) * [Data warehouse](https://en.wikipedia.org/wiki/Data_warehouse" \o "Data warehouse) | * Custom applications databases * Packaged application databases |  | * ETL * EAI * SOA |
| Analytics and [business intelligence](https://en.wikipedia.org/wiki/Business_intelligence" \o "Business intelligence) | * Transactional reporting * Operational analytics * Business analytics * Predictive analytics * Prescriptive analytics * Streaming analytics * Data science and advanced analytics * NLP | * Transactional reporting data access * Operational reporting data access * Analytical reporting data access * Analytical dashboard data access * Operational dashboard data access * [Data mining](https://en.wikipedia.org/wiki/Data_mining" \o "Data mining) | * Real-time dashboards * In-memory analytics * Statistical analysis * [Predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics" \o "Predictive analytics) | * ETL * EAI * TDS * Operational data store * Data mart |
| [Master data management](https://en.wikipedia.org/wiki/Master_data_management" \o "Master data management) | * Master data hub | * Master data replication * Master data services * Master data synchronization |  | * [Change data capture](https://en.wikipedia.org/wiki/Change_data_capture" \o "Change data capture) * EAI * STD |
| [Data modeling](https://en.wikipedia.org/wiki/Data_modeling" \o "Data modeling) | * [Dimensional data modeling](https://en.wikipedia.org/wiki/Dimensional_modeling" \o "Dimensional modeling) * [E-R data modeling](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model" \o "Entity–relationship model) | * Modeling standards * Naming conventions |  |  |
| [Artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence" \o "Artificial intelligence) | * Decision management * [Speech recognition](https://en.wikipedia.org/wiki/Speech_recognition" \o "Speech recognition) * Text analytics and NLP * [Natural language generation](https://en.wikipedia.org/wiki/Natural_language_generation" \o "Natural language generation) * Classic machine learning * Deep learning * [Robotic process automation](https://en.wikipedia.org/wiki/Robotic_process_automation" \o "Robotic process automation) * Image and video analysis |  |  |  |

Some additional examples of architectural patterns:

* [Blackboard system](https://en.wikipedia.org/wiki/Blackboard_(design_pattern)" \o "Blackboard (design pattern))
* [Broker pattern](https://en.wikipedia.org/wiki/Broker_pattern" \o "Broker pattern)
* [Event-driven architecture](https://en.wikipedia.org/wiki/Event-driven_architecture" \o "Event-driven architecture)
* [Implicit invocation](https://en.wikipedia.org/wiki/Implicit_invocation" \o "Implicit invocation)
* [Layers](https://en.wikipedia.org/wiki/Layer_(object-oriented_design)" \o "Layer (object-oriented design))
* [Hexagonal architecture](https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)" \o "Hexagonal architecture (software))
* [Microservices](https://en.wikipedia.org/wiki/Microservices" \o "Microservices)
* [Action–domain–responder](https://en.wikipedia.org/wiki/Action%E2%80%93domain%E2%80%93responder" \o "Action–domain–responder),
  + [Model–view–controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller" \o "Model–view–controller)
  + [Presentation–abstraction–control](https://en.wikipedia.org/wiki/Presentation%E2%80%93abstraction%E2%80%93control" \o "Presentation–abstraction–control)
  + [Model–view–presenter](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter" \o "Model–view–presenter)
  + [Model–view–viewmodel](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel" \o "Model–view–viewmodel)
* [Entity component system](https://en.wikipedia.org/wiki/Entity_component_system" \o "Entity component system)
* [Entity-control-boundary](https://en.wikipedia.org/wiki/Entity-control-boundary" \o "Entity-control-boundary)
* [Multitier architecture](https://en.wikipedia.org/wiki/Multitier_architecture" \o "Multitier architecture) (often three-tier or n-tier)
* [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming" \o "Object-oriented programming)
* [Naked objects](https://en.wikipedia.org/wiki/Naked_objects" \o "Naked objects)
* [Operational data store](https://en.wikipedia.org/wiki/Operational_data_store" \o "Operational data store) (ODS)
* [Peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer" \o "Peer-to-peer)
* [Pipe and filter architecture](https://en.wikipedia.org/wiki/Pipe_and_filter_architecture" \o "Pipe and filter architecture)
* [Service-oriented architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture" \o "Service-oriented architecture)
* [Space-based architecture](https://en.wikipedia.org/wiki/Space-based_architecture" \o "Space-based architecture)
* [Distributed hash table](https://en.wikipedia.org/wiki/Distributed_hash_table" \o "Distributed hash table)
* [Publish–subscribe pattern](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern" \o "Publish–subscribe pattern)
* [Message broker](https://en.wikipedia.org/wiki/Message_broker" \o "Message broker)
* [Hierarchical model–view–controller](https://en.wikipedia.org/wiki/Hierarchical_model%E2%80%93view%E2%80%93controller" \o "Hierarchical model–view–controller)

1. **Stack memory vs Heap memory طبعا زي م احنا كلنا عارفين اننا لما بنعرف متغير في الدوت نت بيتم تخصيص جزءًا من الذاكرة في ذاكرة الوصول العشوائي RAM. تحتوي هذه الذاكرة على ثلاثة أشياء: اسم المتغير ونوع بيانات المتغير وقيمة المتغير ولكن بناء على نوع البيانات يتحدد المكان الذي سيتم فيه التخصيص. يوجد عندنا نوعان من تخصيص الذاكرة وهما Stack memory vs Heap memory ومن هنا تعالى نعرف اهم بعض الفروقات بينهم 1.     ال Stack هو عبارة عن مصفوفة وبذلك يتم تخصيص الذاكرة في كتل متجاورة ويتم الاضافة او الحذف LIFO (Last In First Out) ال Heap هو عبارة عن منطقة من الذاكرة حيث يتم تخصيص أجزاء للتخزين بترتيب عشوائي ويتم الاضافة والحذف باي ترتيب 2.     بالنسبة لتخصيص الذاكرة وإلغاؤها في ال Stack يتم بطريقة تلفائية بواسطة ال Compiler اما في ال Heap فيتم التخصيص بواسطة المبرمج و الالغاء يتم بواسطة المبرمج و GC 3.     يتم تخزين ال Value Types في ال Stack وال Reference Types في ال Heap ال Value Types تكون من النوع Struct ال Reference Types تكون من النوع Class 4.     في حالة اننا بنستخدم ال multi-threading البيانات اللى في ال Stack لايكون متاح استخدامها إلا فقط لل Owner Thread وبذلك يكون Thread safe اما بالنسبة لل Heap يكون متاح للكل All Threads وبذلك يكون Not Thread safe 5.     طبعا في ال Accessing time هنلاقي ال Stack اسرع من ال Heap 6.     ال Stack is fixed-size ولكن ال Heap is resizing is possible 7.     ال Stack is linear لكن ال Heap is hierarchical 8.     ال memory shortage هو المشكلة الرئيسية في ال stack اما في ال Heap فمشكلته الرئيسية هي Memory fragmentation 9.     ال Heap يكون Dynamic Memory Allocation اي انه Mutable اما ال Stack يكون Static Memory Allocation اي انه Immutable وهنا لازم نفرق انه ليس مخصص لل static data**

**5.**

* **[2أنماط التصميم الإنشائية Creational Design Patterns](https://wiki.hsoub.com/Design_Patterns" \l ".D8.A3.D9.86.D9.85.D8.A7.D8.B7_.D8.A7.D9.84.D8.AA.D8.B5.D9.85.D9.8A.D9.85_.D8.A7.D9.84.D8.A5.D9.86.D8.B4.D8.A7.D8.A6.D9.8A.D8.A9_Creational_Design_Patterns)**
  + **[2.1نمط أسلوب المصنع Factory Method](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A3.D8.B3.D9.84.D9.88.D8.A8_.D8.A7.D9.84.D9.85.D8.B5.D9.86.D8.B9_Factory_Method)**
  + **[2.2نمط المصنع المجرد Abstract Factory](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D8.B5.D9.86.D8.B9_.D8.A7.D9.84.D9.85.D8.AC.D8.B1.D8.AF_Abstract_Factory)**
  + **[2.3نمط الباني Builder](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.A8.D8.A7.D9.86.D9.8A_Builder)**
  + **[2.4نمط النموذج الأولي Prototype](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.86.D9.85.D9.88.D8.B0.D8.AC_.D8.A7.D9.84.D8.A3.D9.88.D9.84.D9.8A_Prototype)**
  + **[2.5نمط المفردة Singleton](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D9.81.D8.B1.D8.AF.D8.A9_Singleton)**
* **[3أنماط التصميم الهيكلية](https://wiki.hsoub.com/Design_Patterns" \l ".D8.A3.D9.86.D9.85.D8.A7.D8.B7_.D8.A7.D9.84.D8.AA.D8.B5.D9.85.D9.8A.D9.85_.D8.A7.D9.84.D9.87.D9.8A.D9.83.D9.84.D9.8A.D8.A9)**
  + **[3.1نمط المحوِّل Adapter](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D8.AD.D9.88.D9.90.D9.91.D9.84_Adapter)**
  + **[3.2نمط الجسر Bridge](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.AC.D8.B3.D8.B1_Bridge)**
  + **[3.3نمط المُركَّب Composite](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D9.8F.D8.B1.D9.83.D9.8E.D9.91.D8.A8_Composite)**
  + **[3.4نمط المُزخرِف Decorator](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D9.8F.D8.B2.D8.AE.D8.B1.D9.90.D9.81_Decorator)**
  + **[3.5نمط الواجهة Facade](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.88.D8.A7.D8.AC.D9.87.D8.A9_Facade)**
  + **[3.6نمط وزن الذبابة Flyweight](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D9.88.D8.B2.D9.86_.D8.A7.D9.84.D8.B0.D8.A8.D8.A7.D8.A8.D8.A9_Flyweight)**
  + **[3.7نمط الوكيل Proxy](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.88.D9.83.D9.8A.D9.84_Proxy)**
* **[4أنماط التصميم السلوكية](https://wiki.hsoub.com/Design_Patterns" \l ".D8.A3.D9.86.D9.85.D8.A7.D8.B7_.D8.A7.D9.84.D8.AA.D8.B5.D9.85.D9.8A.D9.85_.D8.A7.D9.84.D8.B3.D9.84.D9.88.D9.83.D9.8A.D8.A9)**
  + **[4.1نمط سلسلة المسؤولية Chain of Responsibility](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.B3.D9.84.D8.B3.D9.84.D8.A9_.D8.A7.D9.84.D9.85.D8.B3.D8.A4.D9.88.D9.84.D9.8A.D8.A9_Chain_of_Responsibility)**
  + **[4.2نمط الأمر Command](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.A3.D9.85.D8.B1_Command)**
  + **[4.3نمط المكرِّر Iterator](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D9.83.D8.B1.D9.90.D9.91.D8.B1_Iterator)**
  + **[4.4نمط الوسيط Mediator](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.88.D8.B3.D9.8A.D8.B7_Mediator)**
  + **[4.5نمط التذكرة Memento](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.AA.D8.B0.D9.83.D8.B1.D8.A9_Memento)**
  + **[4.6نمط المراقِب Observer](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D9.85.D8.B1.D8.A7.D9.82.D9.90.D8.A8_Observer)**
  + **[4.7نمط الحالة State](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.AD.D8.A7.D9.84.D8.A9_State)**
  + **[4.8نمط الخطة Strategy](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.AE.D8.B7.D8.A9_Strategy)**
  + **[4.9نمط أسلوب القالب Template Method](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A3.D8.B3.D9.84.D9.88.D8.A8_.D8.A7.D9.84.D9.82.D8.A7.D9.84.D8.A8_Template_Method)**

**[4.10نمط الزائر Visitor](https://wiki.hsoub.com/Design_Patterns" \l ".D9.86.D9.85.D8.B7_.D8.A7.D9.84.D8.B2.D8.A7.D8.A6.D8.B1_Visitor)**

**10.**

### قواعد البيانات اليدوية  Manual Database

1. 

تلك البيانات المبعثرة في الأرفف والأوراق التي تملأ المكتبات والمخازن تمثل قواعد البيانات اليدوية . و بسبب هذه البعثرة جاءت تصنيفات مهمة ومنطقية لتصنف البيانات حسب علاقتها ببعضها أو حسب بنيتها الرياضية او المنطقية، و بناء على ما ذكرتُ لك تُعتبر الملفات المخزنة في الأرفف و فواتير المبيعات المجدولة إلكترونياً و عناوين الأشخاص في ملف وثائقي أمثلةً لبيانات يدوية.

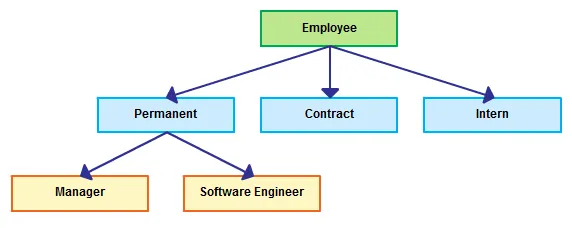
### قواعد البيانات العلائقية Relational Database

هذا النوع من قواعد البيانات من أكثر الأنواع إستخداماً من قبل الإنسان من أجل تنسيق المعلومات، فهو يعتمد على ربط الجداول والمعلومات بطريقة أسهل من أجل سرعة الوصول إلى المعلومات المطلوبة. و هي من الطرق التي يستخدمها العقل البشري كثيراً في محاولات التذكر للأحداث القديمة. علاقة بسيطة قد تجعلك تتذكر أحداثاً كبيرة.  
يمتلك هذا النوع من قواعد البيانات المعلومات مميزات جيدة، كأن يتم إدخالها البيانات مرة واحدة فقط فلا داعي للتكرار. كما أن الجداول الصغيرة يمكن إنشائها وتعديلها بسهولة. فالصغير الواضح ذو العلاقات البينة أسهل في التعديل، إضافة إلى إمكانية إضافة الجداول إلى قاعدة البيانات في أي وقت.

### قواعد البيانات غير العلائقية Non-relational Database

وفى هذا النظام يُنشأ جدول كبير يحتوى على جميع البيانات. كأن كل ما تملك من معلومات في ورقة وحيدة.  
لا يخفى عليك غرابة هذه الطريقة و ربما تخيلت عدة مساوئ لها، ففي هذا النوع من قواعد البيانات تتكرر البيانات بكثرة ، ففي حالة إدخالك لمنتج 10 مرات فسيكتب رقم هاتف المورد مثلاً 10 مرات كذلك!! ولا يخفي عليك عند التعديل سيُعدل الرقم كذل 10 مرات أيضاً!!

### قواعد بيانات ذات الشكل هرمي Hierarchy Database

1. 

وتعتمد هذه القاعدة على مبدأ التسلسل الهرمي في العمل . حيث أنها تقوم بعمل تسلسل من الأصل ، أو الجذر ، حيث أن هذا النظام يبدأ في التفرع على شكل أقسام ، ويقوم مبدأ عمله على الوصول إلى البيانات بطريقة متسلسلة ومتفرعة ، وتكون إما من أسفل للأعلى أو من الأعلى للأسفل .

### قواعد البيانات الشبكية Network Database

ظهر هذا النوع من قواعد البيانات في زمن شهرة قواعد البيانات ذات الشكل الهرمي. يعتمد الشكل الهرمي على أن يكون الأب وحيد و له عدة أبناء، و لكن وُجد أن بعض البيانات ترتبط بطريقة عدة أبناء مع عدة آباء و العكس صحيح. ربما ليس من المنطقي تعدد الآباء في الحقيقه لكنه في قواعد البيانات يحدُث  .  
يقتصر النوعان الرابع والخامس على الإحتياجات الكبيرة لأنهما يتطلبان عادة ذواكر بأحجام كبيرة. ولكن رغم ذلك فإنها لها مزايا عديدة، فهي أكثر كفاءة من قواعد البيانات العلائقية ، وتتعامل مع كم كبير جداً من المعلومات ، بإضافة إلى توفير بناء على طريقة تنظيم الملفات التي تتبعها مساحات كبيرة من وسائط لتخزين البيانات Storage Data base.



**الفرق بين الهرمية والشبكية والعلائقية :**يستخدم النموذجان الهرمي والشبكي روابط (**links**) أو مؤشرات (**pointers**) لوصل السجلات (**Records**) ببعضها البعض في النظام ، وتدعى هذه الأنظمة بالأنظمة الساكنة (**static**) أو المتراصة (**monolithic**) لأن السجلات فيها مربوطة ببعضها بشكل فيزيائي من خلال تعاريفها، وتتميز هذه الأنظمة بأنها معقدة العمل وصعبة التعديل، إلا أن سرعة الوصول فيها تغطي عيوبها .

أما في الأنظمة العلائقية فالربط بين السجلات لا يجري فيزيائياً عن طريق المؤشرات، وإنما عن طريق الأسماء الحقيقية للحقول ، كحقول رقم الموظف أو الإسم أو رقم البطاقة، فالسجلات في هذا النظام قابلة للعنونة بالمحتوى (**connect-addressable**) بحيث يجري الوصول إليها بمطابقة قيم البيانات المخزنة مع بعضها.

تضم الأسواق حالياً مالا يقل عن 200 نظام (**Data Base Managements Systems DBMS**) لإدارة قواعد البيانات ، نصفها تقريباً يستخدم اللغة **SQL** (أنظمة علائقية) على مختلف أنواع الحاسبات.

1. 

تلك البيانات المبعثرة في الأرفف والأوراق التي تملأ المكتبات والمخازن تمثل قواعد البيانات اليدوية . و بسبب هذه البعثرة جاءت تصنيفات مهمة ومنطقية لتصنف البيانات حسب علاقتها ببعضها أو حسب بنيتها الرياضية او المنطقية، و بناء على ما ذكرتُ لك تُعتبر الملفات المخزنة في الأرفف و فواتير المبيعات المجدولة إلكترونياً و عناوين الأشخاص في ملف وثائقي أمثلةً لبيانات يدوية.

### قواعد البيانات العلائقية Relational Database

هذا النوع من قواعد البيانات من أكثر الأنواع إستخداماً من قبل الإنسان من أجل تنسيق المعلومات، فهو يعتمد على ربط الجداول والمعلومات بطريقة أسهل من أجل سرعة الوصول إلى المعلومات المطلوبة. و هي من الطرق التي يستخدمها العقل البشري كثيراً في محاولات التذكر للأحداث القديمة. علاقة بسيطة قد تجعلك تتذكر أحداثاً كبيرة.  
يمتلك هذا النوع من قواعد البيانات المعلومات مميزات جيدة، كأن يتم إدخالها البيانات مرة واحدة فقط فلا داعي للتكرار. كما أن الجداول الصغيرة يمكن إنشائها وتعديلها بسهولة. فالصغير الواضح ذو العلاقات البينة أسهل في التعديل، إضافة إلى إمكانية إضافة الجداول إلى قاعدة البيانات في أي وقت.

### قواعد البيانات غير العلائقية Non-relational Database