Architecture influences the design of components, technology, and middleware.

A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. Eg. Google API

SaaS providers host an application and make it available to users through the internet, usually a browser-based interface. Eg. Gmail, Dropbox, Salesforce, or Netflix.

PaaS solutions appeal to developers who want to spend more time coding, testing, and deploying their applications instead of dealing with hardware-oriented tasks such as managing security patches and operating system updates. Eg. Ms.Azure

IaaS providers deploy and manage[pre-configured and virtualized hardware](http://www.computerweekly.com/photostory/2240109268/The-Computer-Weekly-guide-to-Cloud-Computing/2/The-difference-between-Saas-Paas-and-Iaas" \t "_blank) and enable users to spin up virtual machines or computing power without the labor-intensive server management or hardware investments. Eg. Amazon Web Services offers IaaS through the [Elastic Compute Cloud](https://aws.amazon.com/ec2/" \o "" \t "_blank)

Tier is a physical unit, where the code / process runs. E.g.: client, application server, database server; Layer is a logical unit, how to organize the code. E.g.: presentation (view), controller, models, repository, data access.

Tiers represent the physical separation of the presentation, business, services, and data functionality of your design across separate computers and systems. Layers are the logical groupings of the software components that make up the application or service. They help to differentiate between the different kinds of tasks performed by the components, making it easier to create a design that supports reusability of components.

Enterprise applications usually involve persistent data. Eg. SIAK, billing system, ERP.

Not Enterprise application : word processors, telephone switches, operating systems, compilers, and games.

Response time is the amount of time it takes for the system to process a request from the outside.

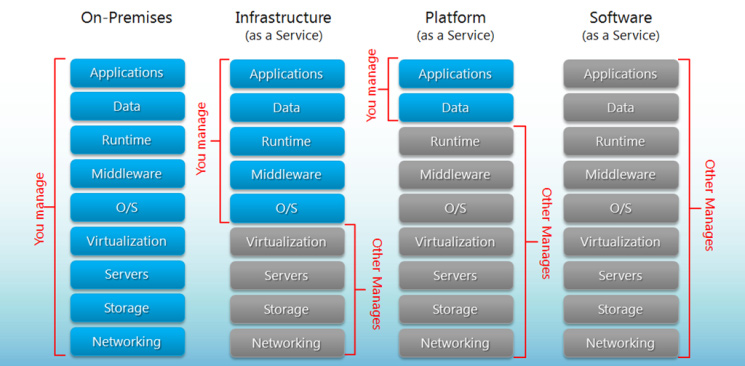
Responsiveness is about how quickly the system acknowledges a request as opposed to processing it.

Latency is the minimum time required to get any form of response, even if the work to be done is nonexistent.

Throughput is how much stuff you can do in a given amount of time. For enterprise applications a typical measure is transactions per second (tps).

Load is a statement of how much stress a system is under, which might be measured in how many users are currently connected to it.

Efficiency is performance divided by resources. A system that gets 30 tps on two CPUs is more efficient than a system that gets 40 tps on four identical CPUs.



Scalability is a measure of how adding resources (usually hardware) affects performance.

Pattern is a particular solution, one that's both common and effective in dealing with one or more recurring problems.

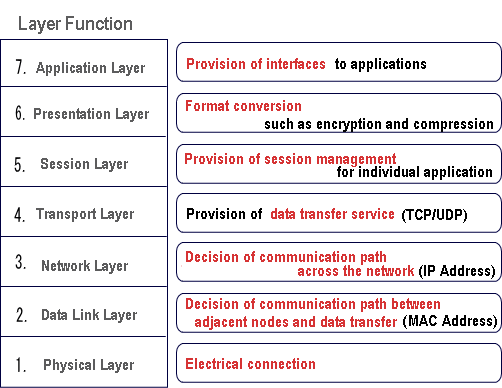
MVC Pattern is used to separate application's concerns.

* Model - represents an object . It can also have logic to update controller if its data changes.
* View - represents the visualization of the data that model contains.
* Controller - acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Implementation

a *Student* object acting as a model. *StudentView* will be a view class which can print student details on console and *StudentController*is the controller class responsible to store data in *Student* object and update view *StudentView* accordingly.

A Layer is a design construct. It is implemented by any number of classes or modules that behave like they are all in the same layer.



Benefits:

* You can understand a single layer as a coherent whole without knowing much about the other layers.
* You can substitute layers with alternative implementations of the same basic services.
* You minimize dependencies between layers.
* Layers make good places for standardization.

Drawbacks:

* Layers encapsulate some, but not all, things well.
* Extra layers can harm performance.

Three layers

* knows about handling http requests and rendering HTML, [presentation layer]
* a business logic layer that contains validations and calculations, [domain layer]
* and a data access layer that sorts out how to manage persistent data in a database or remote services. [data layer]

JavaBeans are classes that encapsulate many objects into a single object (the bean).

Java EE = EJB + Servlet

Application server: HTTP Server, EJB Server, JDBC, JNDI, Servlets, JSP, RMI

Spring calls itself lightweight because you don't need all of Spring to use part of it. For example, you can use Spring JDBC without Spring MVC.

Spring provides various module for different purpose, you can just inject dependency according to your required module. That is you don't need to download or inject all dependencies or all jars to use particular module.

If you want to run a Java EE application, you can't just create a small application that will run on its own. You will need a Java EE application server to run your application, such as Glassfish, WebLogic or WebSphere.

Spring Framework Goals:

* Make J2EE easier to use
* *Enhance productivity compared to “traditional” J2EE approaches*
* Provide the *Inversion of Control* solution
* “Non-invasive” framework: Application code has minimal or *no* dependency on Spring APIs; More power to the POJO
* Fully portable across application servers
* Facilitate unit testing

Module in Spring Framework:

* Core - Inversion of Control (IoC) and Dependency Injection
* AOP - Aspect-oriented programming
* DAO - Data Access Object support, transaction management, JDBC-abstraction
* ORM - Object Relational Mapping data access, integration layers for JPA, JDO, Hibernate, and iBatis
* MVC - Model-View-Controller implementation for web-applications
* Remote Access, Authentication and Authorization, Remote Management, Messaging Framework, Web Services, Email, Testing, …

Example Spring Framework Spring JDBC, Spring MVC, Spring ORM.

We use Maven as our build manager for java project. Add the following dependency on pom.xml file:

Interface Segregation Principle: Clients should not be forced to depend on methods that they do not use. Eg. We have a Manager class which manages the workers. And we have 2 types of workers and they need a daily launch break to eat. But now some robots came in the company they work as well , but they don't eat so they don't need a launch break. One on side the new Robot class need to implement the IWorker interface because robots works. On the other side, the don't have to implement it because they don't eat.

Spring MVC is a Web Application development framework and is also a module of Spring Framework. It needs you to provide a lot of configurations to work properly. Spring Boot provides auto-configuration for your Spring applications. Spring Boot is not an application server only embeds a servlet container, like tomcat. (automation configuration, starter dependencies, command line interface).

Separating presentation dan model:

1. Mengizinkan kita membuat presentasi yang banyak, sementara modelnya tetap.
2. Presentation dan view adalah 2 hal yang berbeda. View => isu UI dan layouting. Presentation => business policies dan database interactions. Library yang digunakan beda-beda. Programmer/tim pengembangnya pun bisa dibedakan.
3. Model lebih mudah ditest daripada view, pemisahan ini memudahkan test model

Pattern in organizing domain logic:

Transaction Script : domain logic is primarily organized by the transactions that you carry out with the system. Eg. book hotel room, the logic to check room availability, calculate rates, and update db is found inside the Book Hotel Room procedure. Advantages: most understand, works well with a simple data source layer, obvious how to set the transaction boundaries. Disadvantages: problem increase domain logic increase, duplicated code as several transactions.

Domain Model : An object-oriented way to handle a problem, primarily around the nouns in the domain. Advantages: More options in structuring the code when the business model getting complex. Advantages: It’s “hard” to trace the behaviour, learn OO.

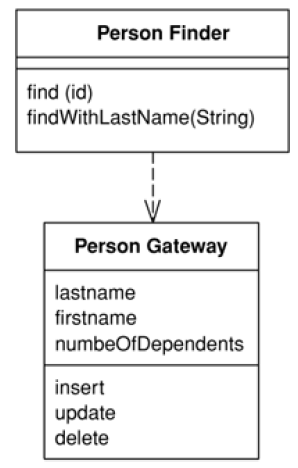
*Table Module* organizes domain logic with one class per table in the database, and a single instance of a class contains the various procedures that will act on the data. Disadvantages : doesn't give you the full power of objects in organizing complex logic, can’t have direct instance-to-instance relationships, and polymorphism doesn't work well.

*Service Layer* is a pattern for organizing business logic (domain and application logic).

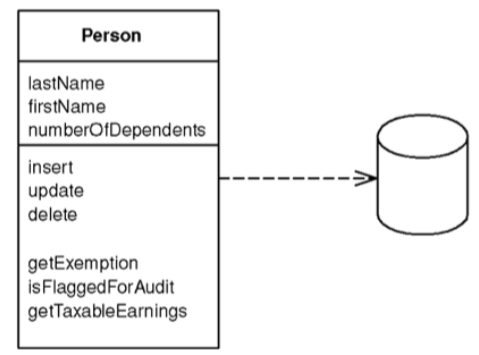
Separate SQL access from the domain logic and place it in separate classes.

Pattern:

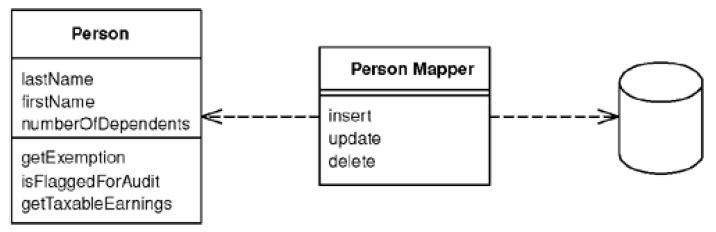
* Row data gateway: *An object that acts as a Gateway to a single record in a data source. (used in Transaction Script )*

**

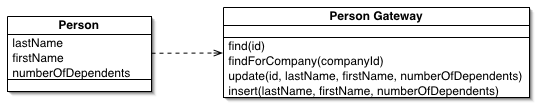
* Active record: *An object that wraps a row in a database table or view, encapsulates the database access and adds domain logic on that data (used in Transaction Script )*

**

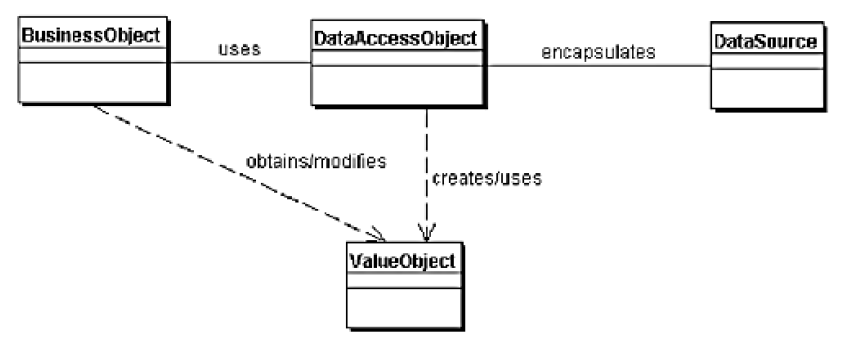
* Data mapper : *moves data between objects and a database* ***while keeping them independent of each other*** *and the mapper itself. (used in Domain Model ). you can ignore the database, both in design and in the build and testing process.*

**

* Table data gateway: *one instance handles all the rows in the table. (used in Transaction Script )*

**

* DAO returns domain objects (DO) that represent instances of data entities.



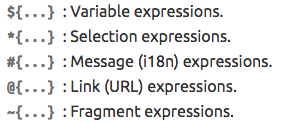
**Input controller patterns:**

* Page Controller : *An object that handles* ***a request*** *for* ***a specific page*** *or action. (controller logic simple)*
* Front Controller : *that* ***handles all requests***

***Web handler pulls just enough information from the URL and the request to decide what kind of action to initiate and then delegates to a command to carry out the action***

**View patterns:**

* Template View: *Renders information into HTML by* ***embedding markers*** *in an HTML page*

**

* Transform View : *processes domain data* ***element by element*** *and transforms it into HTML,* ***organized around separate transforms for each kind of input element***

