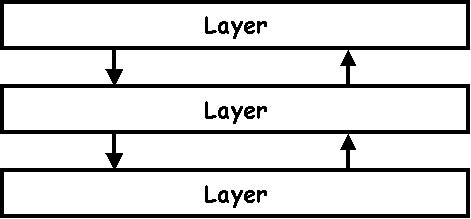
Explain layered system with example

Top of Form

Layered Systems use layers to separate different units of functionality. Each layer only communicates with the layer above and the layer below. Each layer*uses* the layer below to perform its function. Communication happens through predefined, fixed interfaces.

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. That means that they only communicate with classes in layers immediately above or below their layer and with themselves.



**Examples**

* Operating Systems.
* Communication Protocols.

**Where does it come from?**

The first use of layers in software architecture is in [The structure of the 'THE'-multiprogramming system](http://www.cs.utexas.edu/users/EWD/ewd01xx/EWD196.PDF) by E.W. Dijkstra (1968).

**When should you use it?**

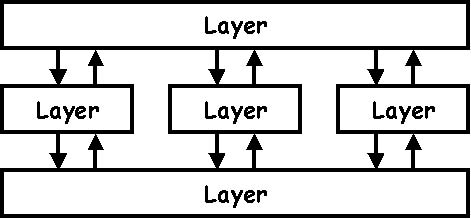
The layered system is ideal when your system has different levels of functionality. The function you want your system to perform is actually performed at the lowest level, or layer. But you do not want to call the functions at the lowest level directly all the time, because they are complex to use, they have several different implementations, etc. At the highest level you just want a few simple functions to do the things you want. Let the lower layers sort the rest out.

**How does it work?**

Each layer offers its own kind of functionality. Lower layers are usually more domain specific, closer to the machine's hardware. A higher layer uses a its lower layer to perform its function. It *requires* its lower layer. It is even allowed for a layer to use other layers that are lower or higher than its immediate neighbor, but these should remain exceptions. They are called **bridges** and should be properly documented.

Each layer contains any number of objects. These objects communicate with other objects in the same layer and in the layers above and below.

Its possible to define multiple layers at the same level. It implies that objects of these layers do not communicate with eachother.



The user calls a function on an object in the upper layer. This object calls functions in the layer below. These functions in turn approach the layer below and the layer above. Etc. Eventually the function is performed and control is returned to the user. All this is usually done sequentially, in the same thread.

**Problems**

* It can be hard to specify and maintain fixed interfaces between the layers.
* Layers are sometimes distinguished from Tiers. But the arguments haven't really convinced me. Both layers and tiers make use of their adjacent units. Tiers are often located on different machines, but this is not essential to the architecture.

**Common implementation techniques**

* Layers need no explicit representation. You don't need to create *layer*classes or the like. Just make sure the objects you create only talk to objects in the correct layers. It's a good idea to use the name of the layer in the classes or modules that belong to it.