Terminology

- Architecture
  - An overall blueprint/model describing the structures and properties of a "system”
- Software architecture
  - Captures the gross structure of a system
  - How it is composed of interacting parts
  - How the interactions take place
  - Key properties of the parts
  - Provides a way of analyzing systems at a high level of abstraction!
  - Illuminates top-level design decisions

Software architecture patterns

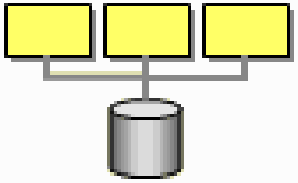
- Are software patterns that offer well-established solutions to architectural problems in software engineering.
- Gives description of the elements and relation type together with a set of constraints on how they may be used.
- express a fundamental structural organization schema for a software system, which consists of subsystems, their responsibilities and interrelations.
- In comparison to design patterns, architectural patterns are larger in scale.

typical architectural patterns (styles)

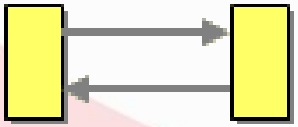
- How can I integrate multiple applications so that they work together and can exchange information?

integration styles for (enterprise) messaging

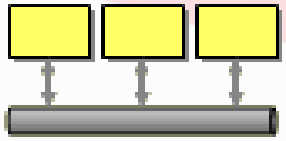
File Transfer



Shared Database



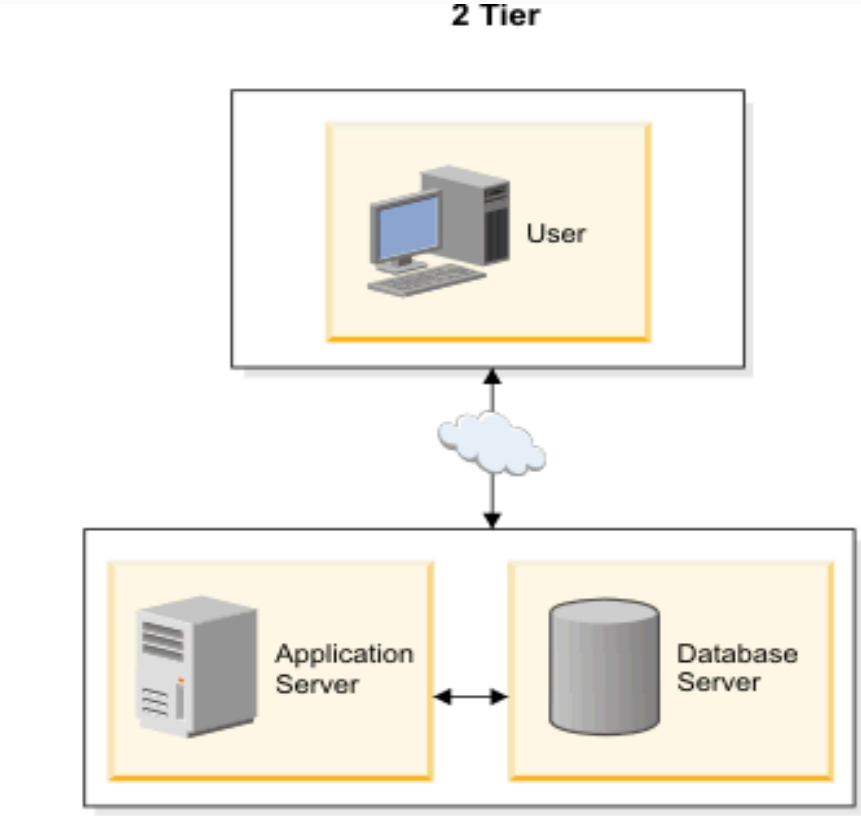
Remote Procedure



Messaging

tiered architecture (layering)

- 2 - tier architecture (traditional client-server)
  - A two-way interaction in a client/server environment, in which the user interface is stored in the client and the data are stored in the server.
  - The application logic can be in either the client or the server.



3 tier architecture

Presentation tier

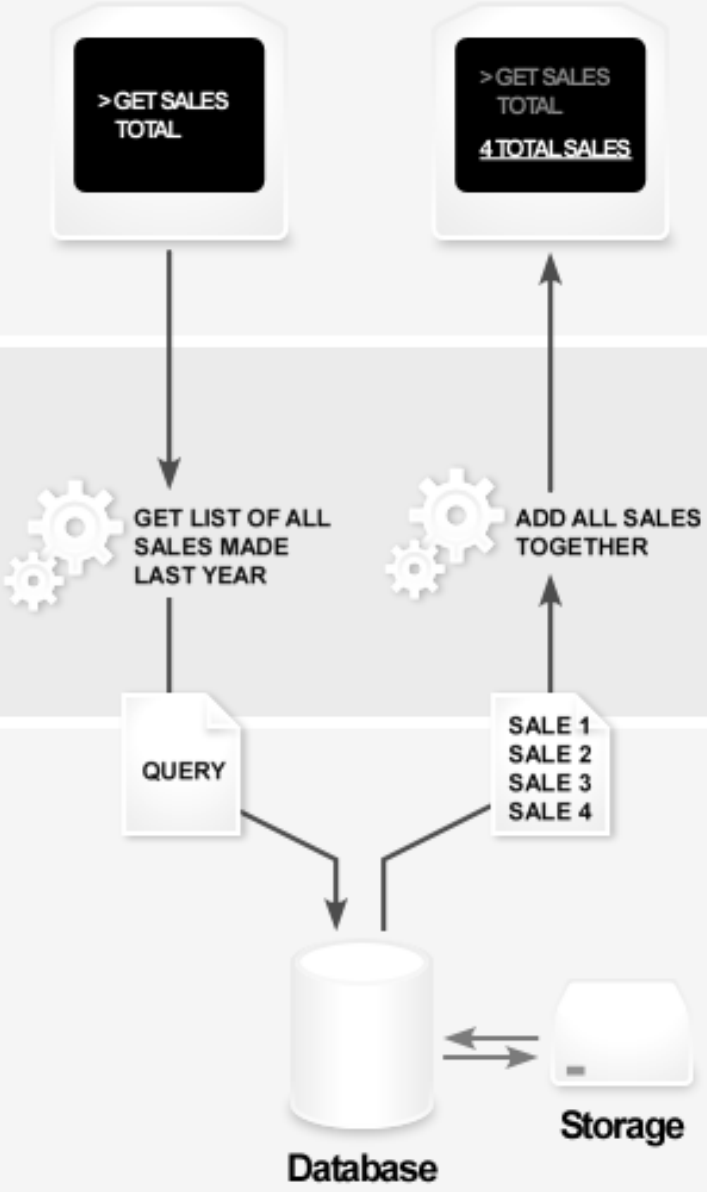
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



model-view-controller (1)

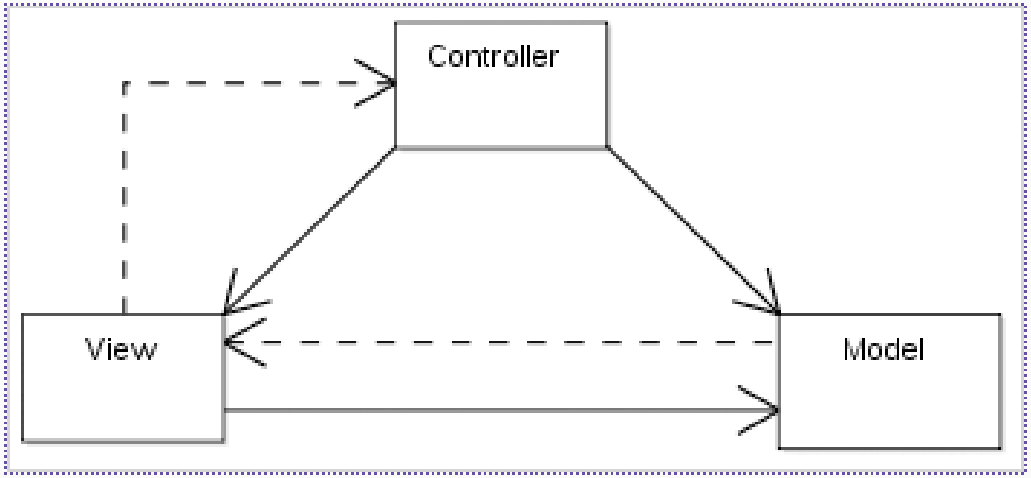
- The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.
- MVC was originally developed to map the traditional input, processing, output roles into the GUI realm:

  Input $\rightarrow$ Processing $\rightarrow$ Output

  Controller $\rightarrow$ Model $\rightarrow$ View

Model View Controller (MVC)

- The intent of MVC is to keep neatly separate objects into one of tree categories
  - Model
    - The data, the business logic, rules, strategies, and so on
  - View
    - Displays the model and usually has components that allows user to edit change the model
  - Controller
    - Allows data to flow between the view and the model
    - The controller mediates between the view and model



Model-View-Controller concept. Note: The solid line represents a direct association, the dashed an indirect association via an observer (for example).

Model

- Responsibilities
  - Provide access to the state of the model
    - getters, toString, other methods that provide info
  - Provide access to the system's functionality
    - changeRoom(int), shootArrow(int)
  - Notify the view(s) that its state has changed

```
// If extending Java's Obervable class, do NOT forget
// to tell yourself your state has changed
super.setChanged();
// Otherwise, the next notifyObservers message will not
// send update messages to the registered Observers
this.notifyObservers();
```

View

- Responsibilities
  - Display the state of the model to users, accept input
  - The model (a.k.a. the Observable) must register the views (a.k.a. Observers) so the model can notify the observers that its state has changed
  - Java's Observer/Observable support provides

```
public void addObserver(Observer o)
// Adds an observer to the set of observers for this
// object, provided that it is not the same as some
// observer already in the set.
```

Controller

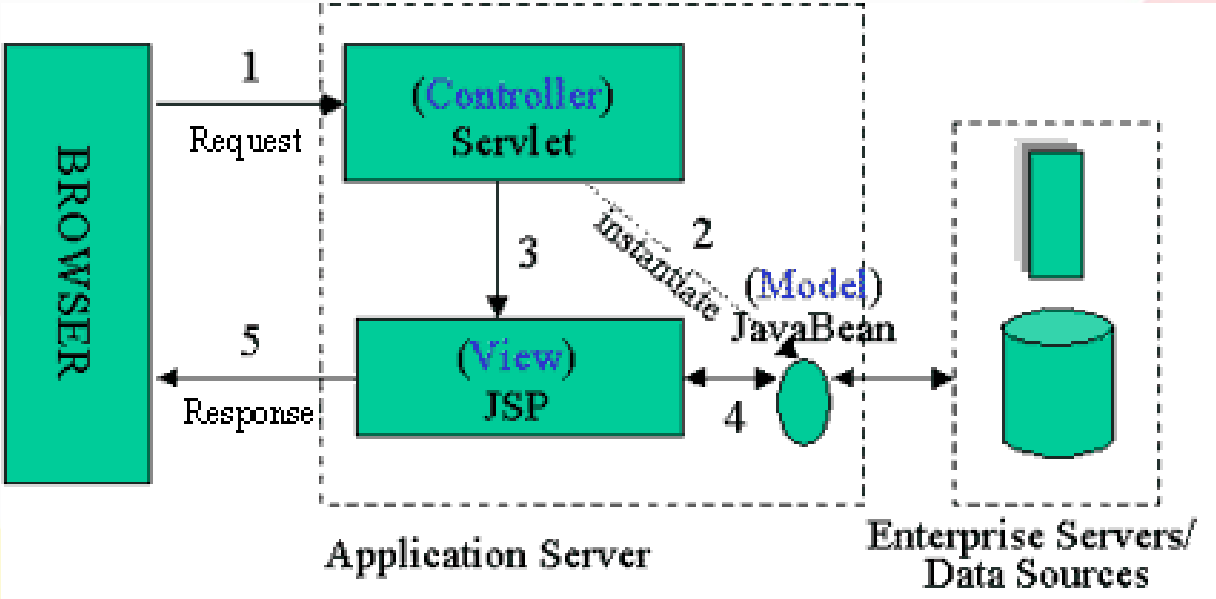
- Responsibilities
  - Respond to user input (events)
    - Button click, key press, mouse click, slider bar change
  - Send messages to the model, which may in turn notify it observers
  - Send appropriate messages to the view
- In Java, controllers are implemented as listeners
  - An *ActionListener* object and its actionPerformed method is a Controller

Sun says

- Model-View-Controller ("MVC") is the recommended architectural design pattern for interactive applications
- MVC organizes an interactive application into three separate modules:
  - one for the application model with its data representation and business logic,
  - the second for views that provide data presentation and user input, and
  - the third for a controller to dispatch requests and control flow.

Java Server Pages

- Model: Enterprise Beans with data in the DBMS
  - JavaBean: a class that encapsulates objects and can be displayed graphically
- Controller: Servlets create beans, decide which JSP to return
  - Servlet object do the bulk of the processing
- View: The JSPs generated in the presentation layer (the browser)



MVC Benefits

- The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.
- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one doesn't affect other modules
  - can develop in parallel once you have the interfaces
- Multiple views
  - spreadsheets, powerpoint, file browsers, games, Eclipse, UML reverse engineering, ….

Two Views of MVC