Model-View-Controller

(or Model-View-Presenter)

MVC

- THE classic design pattern
- Used for data-driven user applications
- Such apps juggle several tasks:
  - **Loading** and **storing** the **data** – getting it in/out of storage on request
  - **Constructing** the **user interface** – what the user sees
  - **Interpreting user actions** – deciding whether to modify the UI or data
- These tasks are largely independent of each other
- Model, View, and Controller each get one task

Model

talks to data source to retrieve and store data

Which database tables is the requested data stored in?

What SQL query will get me the data I need?

View

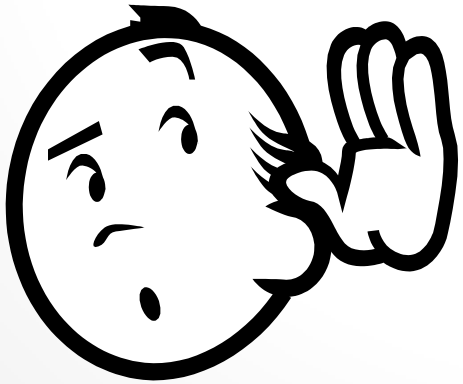
asks model for data and presents it in a user-friendly format

Would this text look better blue or red? In the bottom corner or front and center?

Should these items go in a dropdown list or radio buttons?

Controller

listens for the user to change data or state in the UI, notifying the model or view accordingly



The user just clicked the "hide details" button. I better tell the view.

The user just changed the event details. I better let the model know to update the data.

MVC: Summary

Model

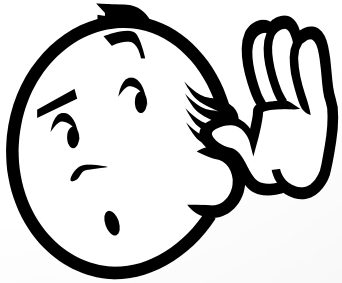
talks to data source to retrieve and store data

View

asks model for data and presents it in a user-friendly format

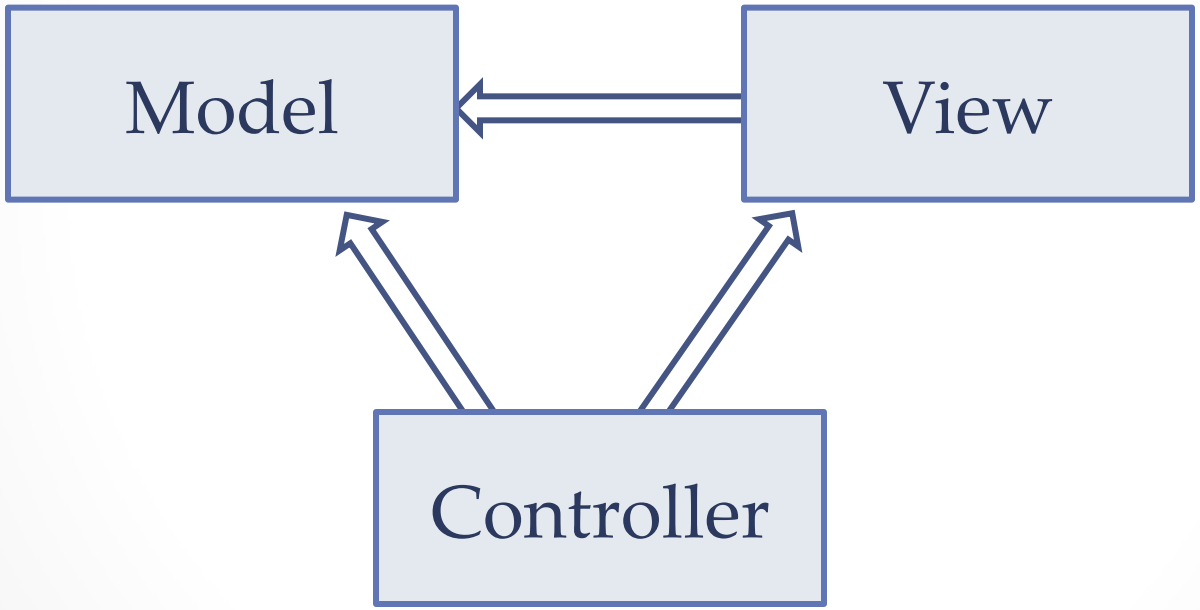
Controller

listens for the user to change data or state in the UI, notifying the model or view accordingly



Communication Flow

Taken from http://msdn.microsoft.com/en-us/library/ff649643.aspx



What do you think are the benefits of MVC?

Benefits of MVC

- Organization of code
  - Maintainable, easy to find what you need
- Ease of development
  - Build and test components independently
- Flexibility
  - Swap out views for different presentations of the same data (ex: calendar daily, weekly, or monthly view)
  - Swap out models to change data storage without affecting user