Please be considerate and refrain from talking with your neighbor until **AFTER** Lecture – Thank You

**FINAL Group Advising Session this WEEK**

– **Thu, Apr 5, 5:00-8:00pm** – Engineering Hall Aud

Go to **schedule.cis.ksu.edu\***@ approx. 4:45 to sign-up

(Click on Group Advising)

**\***Can also schedule an appt w/any adviser through the same site

* **Project 7** available – due **Fri – Apr 13th**
* **Codelab** (Program Design)due **Mon – Apr 9th**
* **THU** Labs **WILL** meet this Week (*Model-View-Controller*)

**Friday 3:30 lab is cancelled** due to Engineering Open House

**…**See announcement in Canvas on options in **making up this lab**

**Today**: **Program Design** using **Model-View- Controller** (MVC) **Architecture**…(not covered in Gaddis Text)

**CIS 501 – Software Arch. & Design**

**Course Description:** Principles and patterns for design and structure of software, development of object-oriented models, examples of software architecture.

**40-20-40 Rule**: **40%** Planning/Design-20% Coding-40% Testing/Debugging

1. **Model:** stores and manipulates data / does calculations
2. **View:** user interface / user interaction (input/output)

* **input/output only** … no processing logic (i.e. Calculations)

1. **Controller:** controls flow of the program

* usually the application (driver) class with main method

**Using *MVC* architecture,** get an integer number from the user and display the square of that number

1. **Model**: store number; method to square the number
2. **View**: method to get number; method to display result
3. **Controller**: *main* method w/calls between Model/View

* To show advantage of MVC, created a ***View\_GUI*** class to change the *interface* (*view*) from *Console* I/O to *GUI* I/O

**MVC Architecture** of a **Connect Four** game (Console I/O)

1. **Model** (Data):holds the game **Board** (2D char array)

…***toString*** used to display current *state* of the board

Store/Manipulate the data only … **NO I/O**

* + Create/Initialize a 2D array for the board (6 X 7)
  + Take a turn (Valid move? Place ‘X’ or ‘O’ at position)
  + Determine if user has four in a row (winner)
  + Determine if board is full (*boolean*)
  + Return *String* representing current **state** of the board

1. **View: IO** **class** handles *Input* & *Output* only

**Input -** where user wants to drop their piece

**Output –** display current **state** of board; “*game over*” results (winner, tie); error messages (*board full, invalid move*)

1. **Controller: ConnectFour** contains **main** method **connecting** the ***model*** and ***view*** classes by creating objects of each class.

* Contains **no *Scanner/input***methods **nor *println*** methods, since this would tie it to a specific “view”