Chapter 5
Software Architecture

Two levels of software design

1. Detailed design

- What operations do we need?
- What data structures?
- What algorithms are we using?
- How is the database going to be organized?
- What does the user interface look like?
- What are the calling sequences?

Two levels of software design

2. Style

- If you were building a house, this design level asks questions like ranch or multi-story? Tudor or Cape Cod? Which direction do the bedroom windows face? Forced-air or hot-water heat? Three bedrooms or four? Open concept floor plan or closed? These questions focus somewhat on details, but they're much more about the style of the house and how you'll be using it, rather than things like 12- or 14-gauge wire for the electrical system or the dimensions of the air conditioning ductwork.

General Architectural Patterns

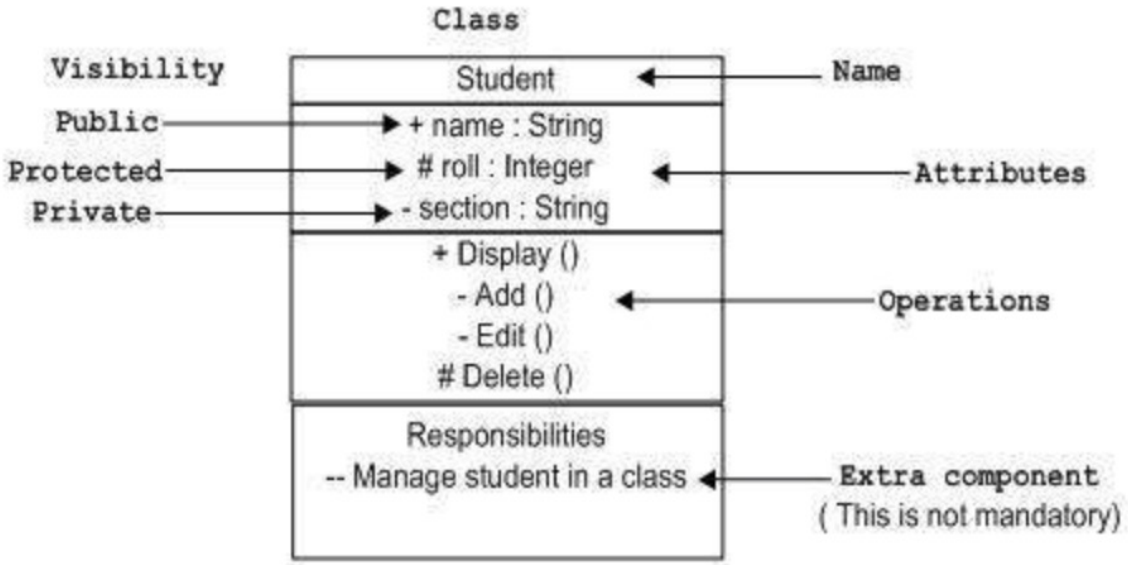
- Software architectures are normally represented as black box *graphs* where graph nodes are *computational structures* and the graph edges are *communication conduits* between the structures.

Unified Modeling Language

- UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

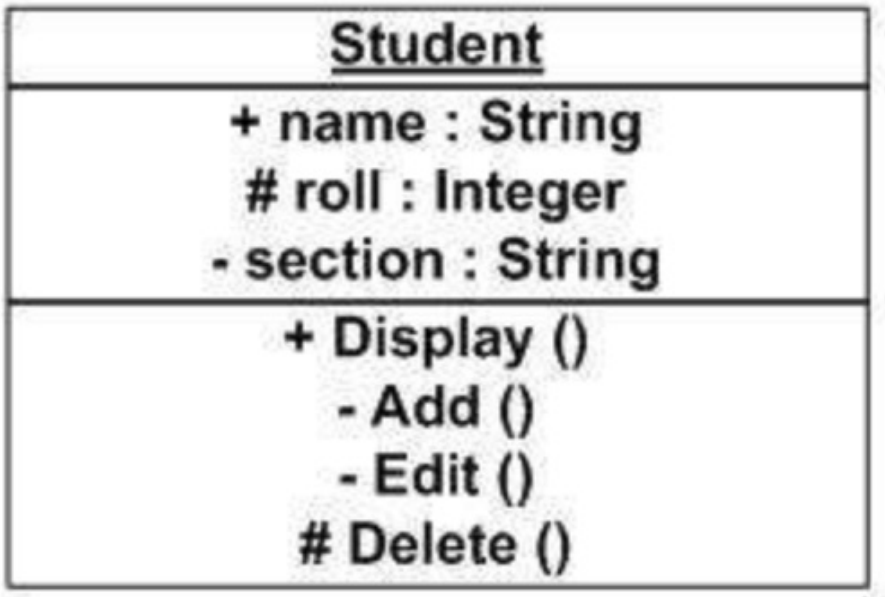
Class Notation

- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.

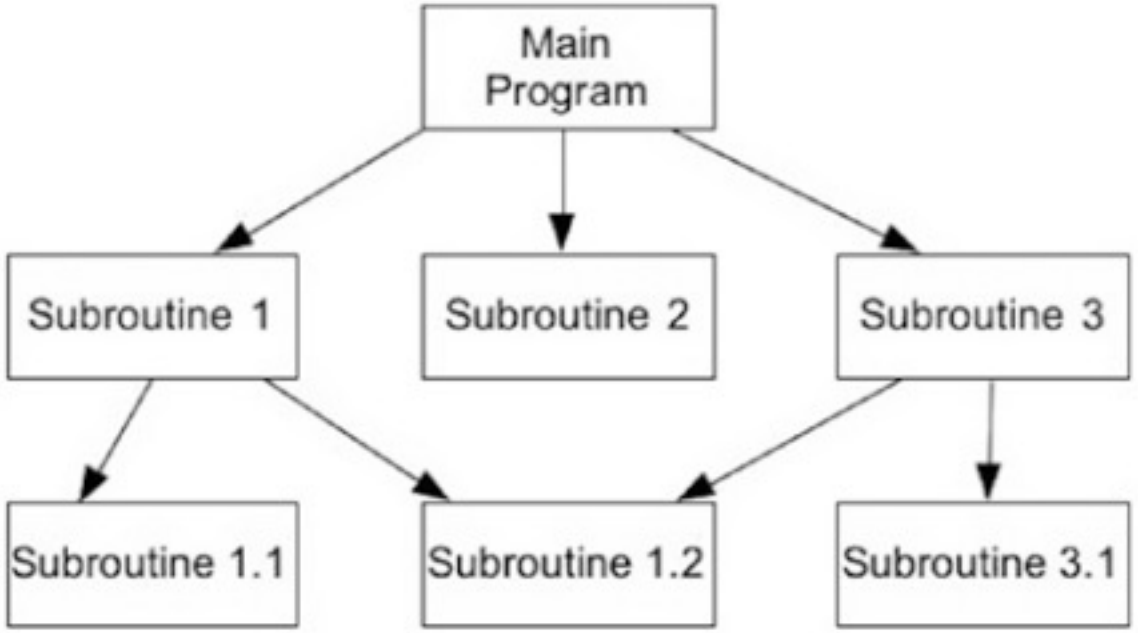


Object Notation

- The *object* is represented in the same way as the class. The only difference is the *name* which is underlined as shown in the following figure.



The Main Program—Subroutine Architectural Pattern



Pipe-and-Filter Architecture

In a pipe-and-filter style architecture, the computational components are called *filters* and they act as transducers that take input, transform it according to one or more algorithms, and then output the result to a communications conduit. The input and outputs conduits are called *pipes*.

Pipe-and-Filter Architecture

- **The Problem:** Given a dictionary of words in English, find all the anagrams in the dictionary. That is, find all the words that are permutations of each other. For example, pots, stop, *and spot are anagrams of each other.*
  - Create a sign for each word in the list by sorting the letters in each word; keep the sign and the word together.
  - Sort the resulting list by the signs; all the anagrams should now be together.
  - Squash the list by putting each set of anagrams on the same line, removing the signs as you do.
- In Unix-speak it looks like this:
  - sign <dictionary.txt | sort | squash >anagrams.txt

An Object-Oriented Architectural Pattern

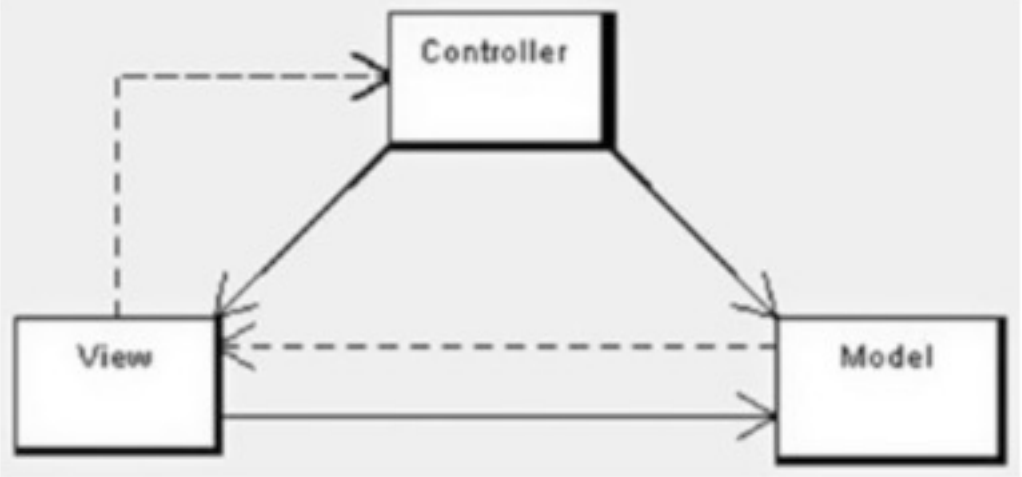
- The **Model-View-Controller (MVC)** architectural pattern is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller. MVC was originally developed to map the traditional input, processing, and output roles of many programs into the GUI realm:

  Input ➤ Processing ➤ Output

  Controller ➤ Model ➤ View

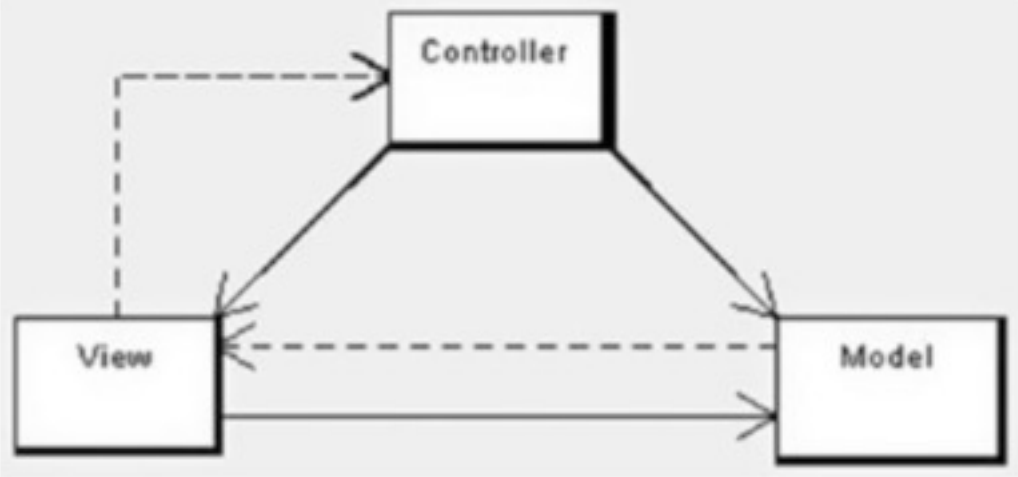
The Model-View-Controller architecture

- The ***controller*** interprets mouse and keyboard inputs from the user and maps these user actions into commands that are sent to the model and/or view to effect the appropriate change.



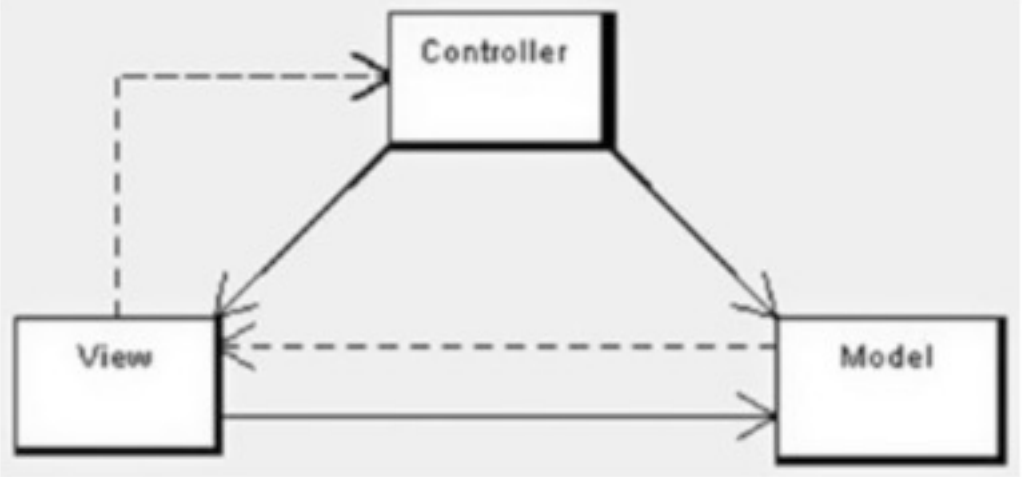
The Model-View-Controller architecture

- The ***model*** manages one or more data elements, responds to queries about its state, and responds to instructions to change state.

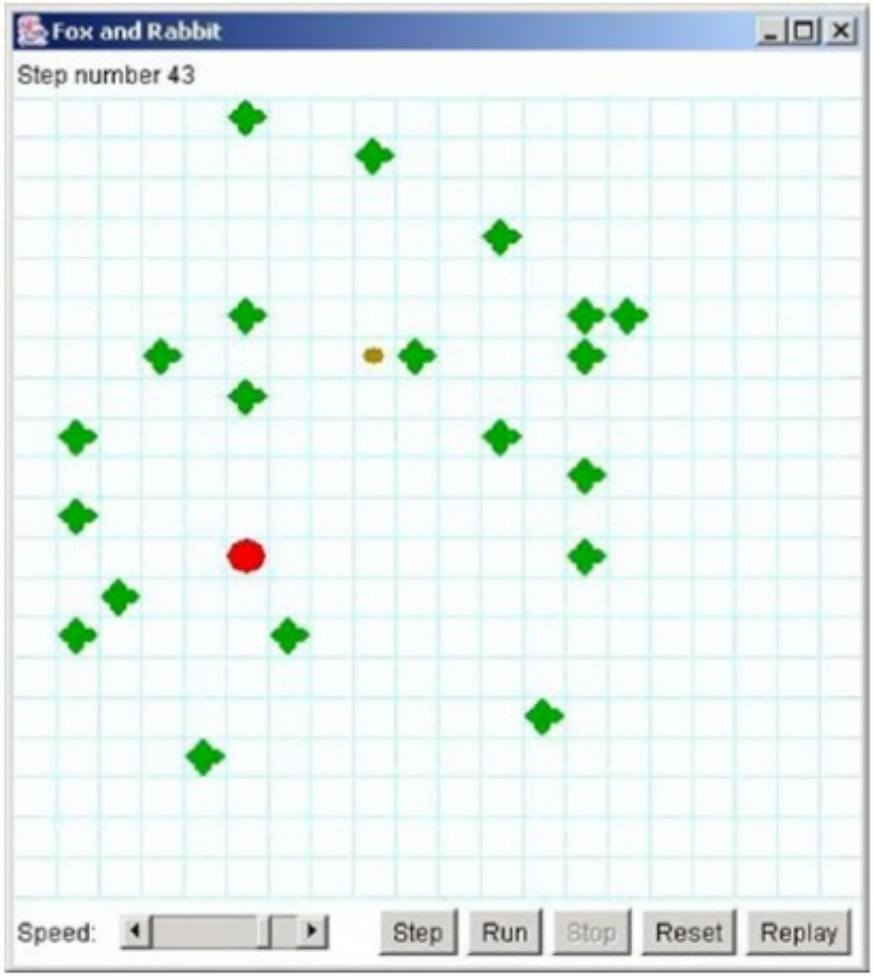


The Model-View-Controller architecture

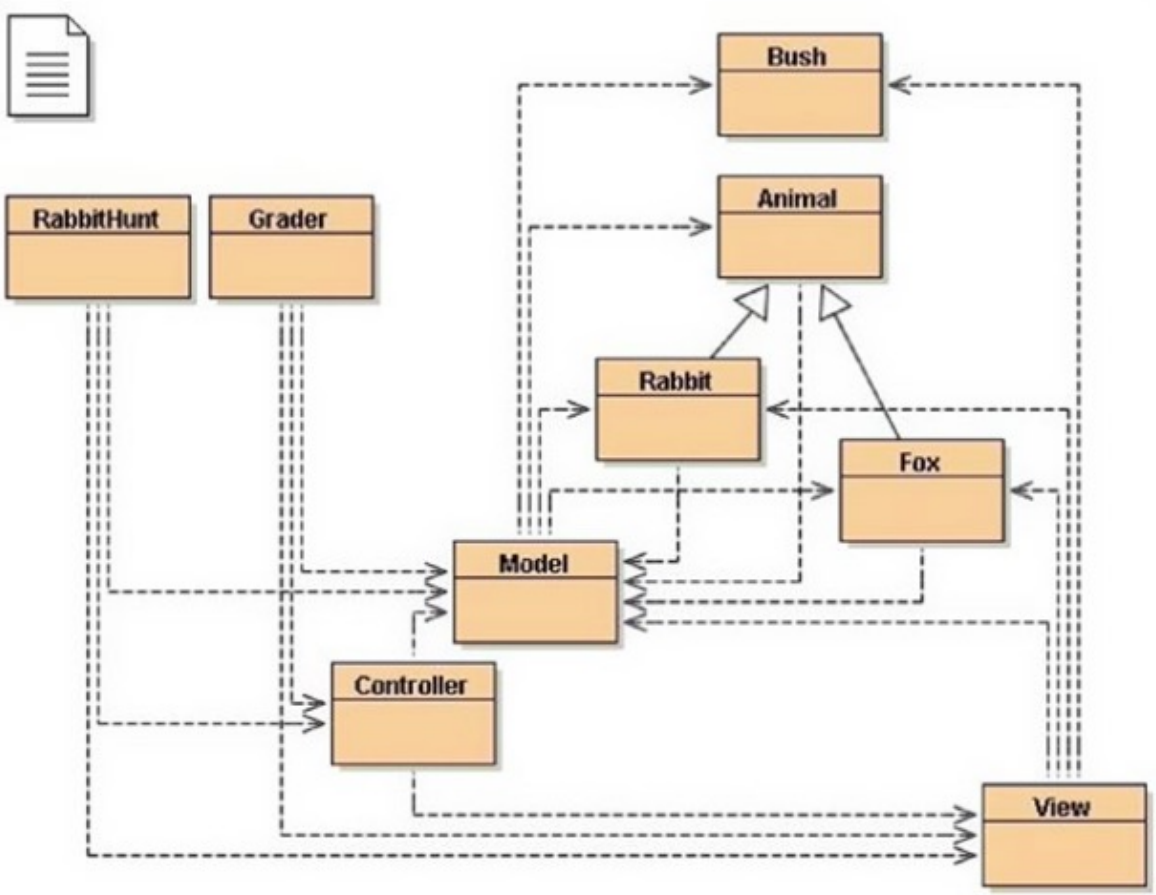
- The ***view*** **or** ***viewport*** manages a rectangular area of the display and is responsible for presenting data to the user through a combination of graphics and text.



A typical fox and rabbit hunt instance

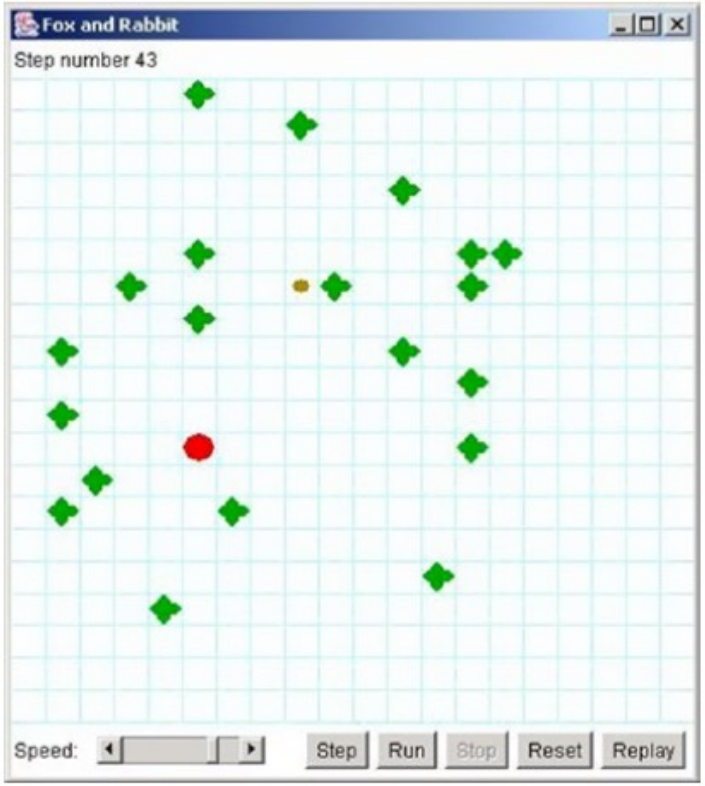


The fox and rabbit hunt class structure



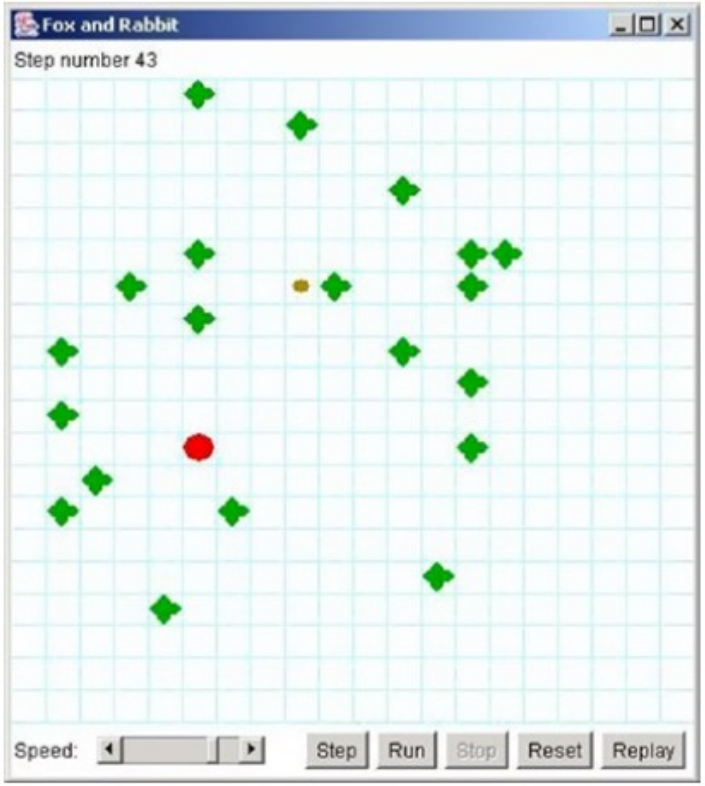
Model

- The **model** represents the rules of the game. It does all the computation and all the work of deciding whose turn it is, what happens during each turn, and whether anyone has won. The model is strictly internal and has practically nothing to do with the other parts of the program.



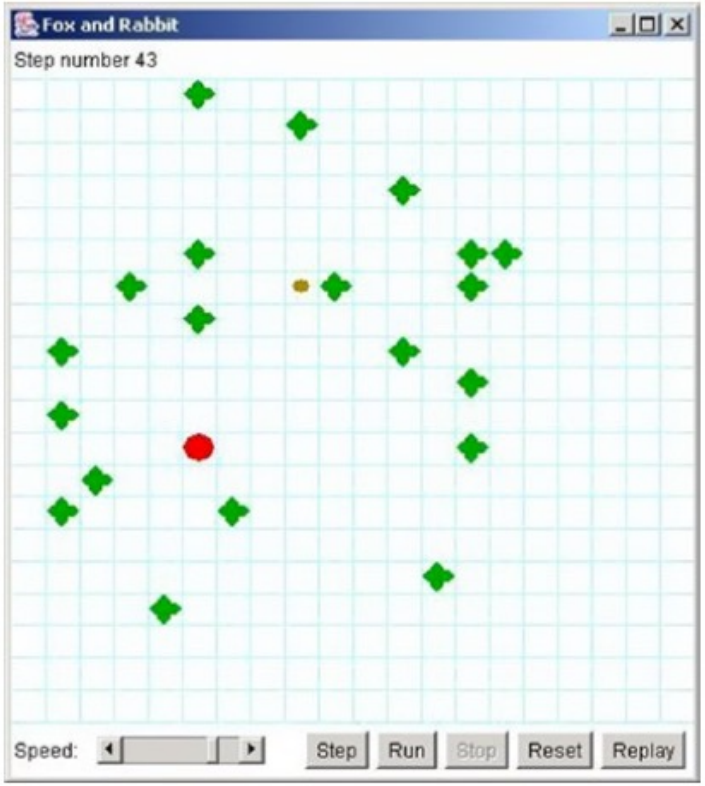
View

- The **view** displays what is going on. It puts an image on the screen so the user can see what's happening. The view is completely passive; it doesn't affect the hunt in any way, it's just a news reporter that gives you a (partial) picture of what is happening inside the model.



Controller

- The **controller** is the part of the program that displays the controls (the five buttons and the speed controls at the bottom of the window). It knows as little as possible about the model and view; it basically tells the model when to go and when to stop.



The Client-Server Architectural Pattern

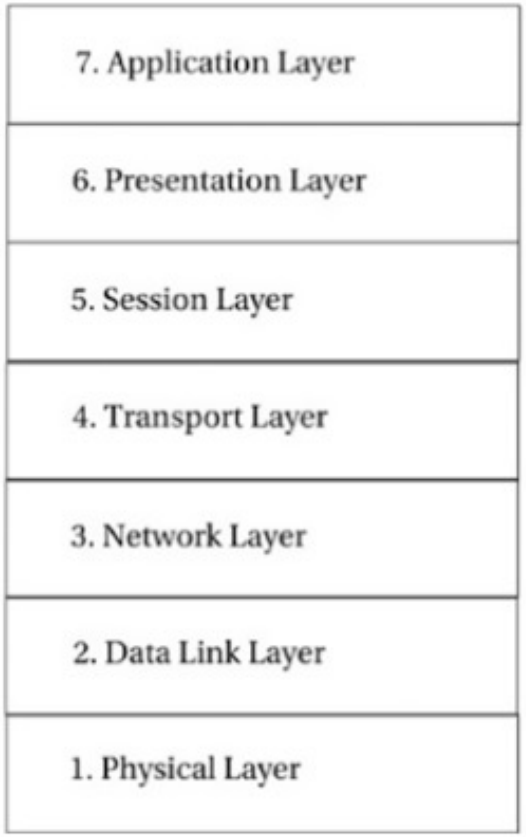
- program is broken up into two different pieces that typically run on two separate computers.
  - A **server** does most of the heavy lifting and computation; it provides services to its clients across a high-bandwidth network.
  - **Clients**, on the other hand, mostly just handle user input, display output, and provide communication to the server.

The Layered Approach

- The layered architectural approach suggests that programs can be structured as a series of layers, much like geologic strata, with a sequence of well-defined interfaces between the layers.

A layered architecture

International Standards Organization (ISO) Open Systems Interconnection (OSI) seven- layer model



The ISO-OSI Layered archictecture

Layer	Protocol
7. Application	HTTP, FTP, telnet
6. Presentation	MIME, SSL
5. Session	Sockets
4. Transport	TCP, UDP
3. Network	IP, IPsec
2. Data Link	PPP, Ethernet, SLIP, 802.11
1. Physical	

Layered Protocols Using the ISO-OSI Architecture

End