8.6.2 Introducing the Model-View-Controller Pattern

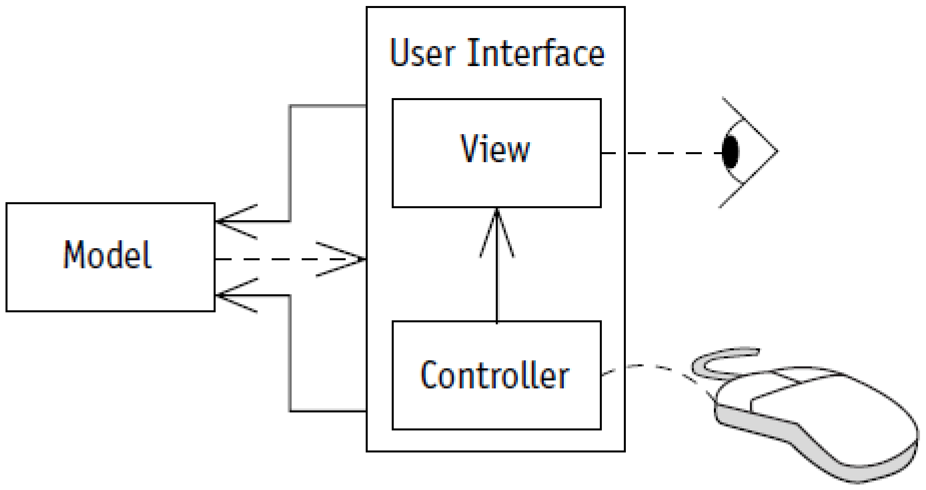
Collaborating classes are also used with modern graphical user interfaces via the Model-View-Controller pattern. This pattern splits a program into three collaborating classes or groups of classes.

- The **model** is responsible for modeling the current problem. For example, the `AlarmClock` class we wrote earlier models the problem of keeping the current time and determining when to ring the alarms, but has little to do with displaying anything to the user.

- The **view** shows relevant information in the model to the user. In an alarm clock program, the view is the class (or group of classes) that show the user what time it is and when the alarms are due to ring. This is information that the view obtains from the model.
- The **controller** is responsible for gathering input from the user and using it to modify the model, for example, by changing the current time or the time when an alarm is due to ring. When the controller changes the model, the view should also change to show the new information.

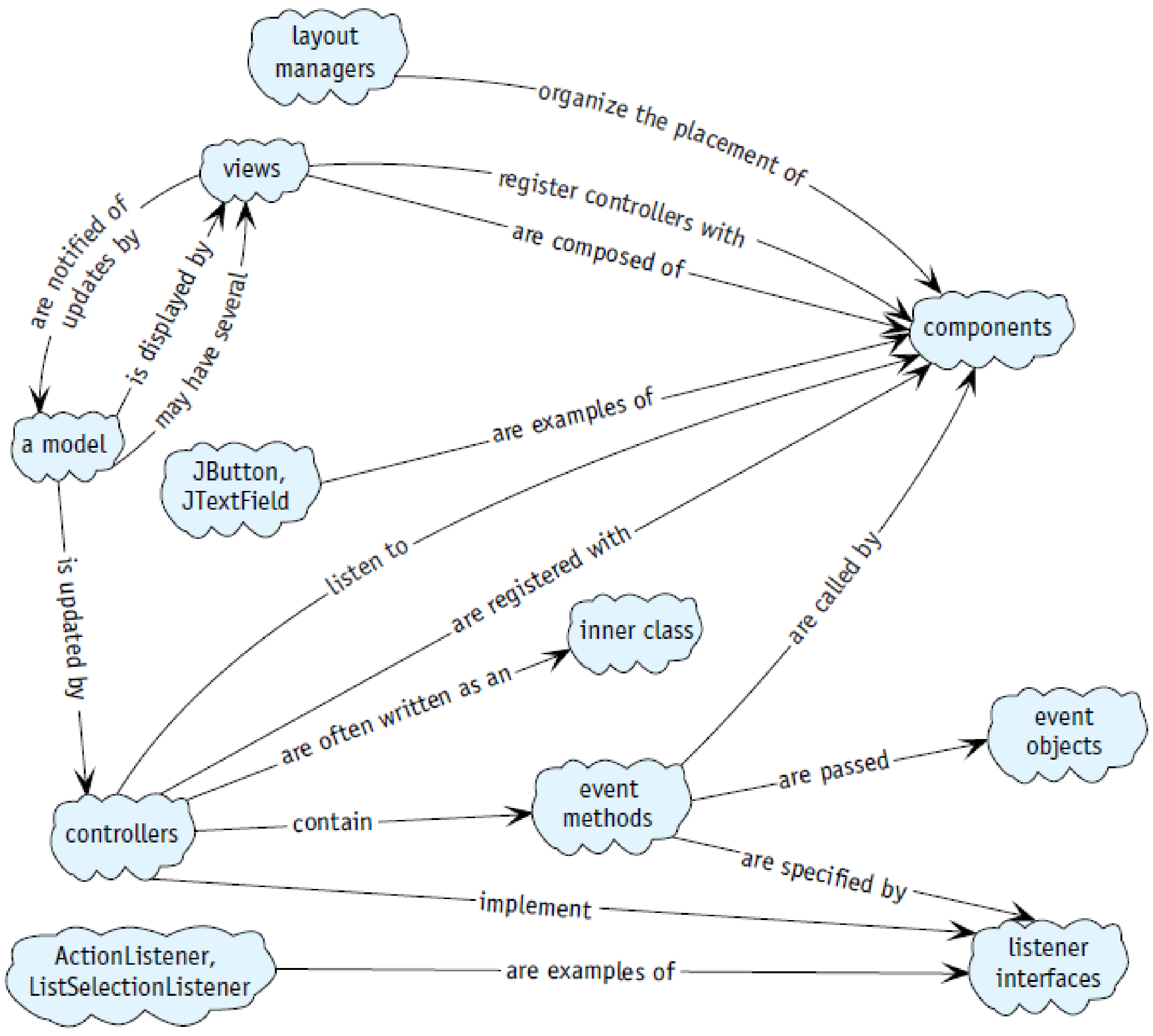
The view and the controller work together closely and are known as the user interface.

The relationships between these three groups of classes are shown in Figure 8-24. The eye represents the user observing the model via the view. The mouse represents the user changing the model via the controller. The arrow between the controller and the view indicates that the controller may call methods in the view, but the view has no need to interact with the controller. The two arrows from the user interface to the model indicate that both the view and the controller will have reason to call methods in the model. The last arrow is dotted to indicate that the model will call methods in the user interface, but in a limited and controlled way.



The interaction of the controller, model, and view may seem complicated at first. However, it follows a standard pattern, which includes the following typical steps, performed in the following order:

- The user manipulates the user interface—for example, enters text in a component.
- The user interface component notifies its controller by calling a method that we write.
- The controller calls a mutator method in the model, perhaps supplying additional information such as text that was entered in the component.
- Inside the mutator method, the model changes its state, as appropriate. Then it calls the view's `update` method, informing the view that it needs to update the information it displays.
- Inside the `update` method, the view calls accessor methods in the model to gather the information it needs to display. It then displays that information.


layout managers
organize the placement of
views
register controllers with
are composed of
components
are notified of updates by
is displayed by
may have several
a model
JButton, JTextField
are examples of
listen to
are registered with
is updated by
inner class
are often written as an
are called by
event objects
are passed
event methods
controllers
contain
are specified by
implement
ActionListener, ListSelectionListener
are examples of
listener interfaces