```
1 public void update( Graphics g )
2 {
3     paint( g );
4 }
```

Figure D.11 Overriding `update` to avoid erasing the canvas on a `repaint`

Recall that, by default, a call to `repaint` clears the component. Sometimes we would prefer to write on the canvas without erasing it. To do this, we override the `update` method for the new canvas class, as shown in Figure D.11.

If there are no threads, `repaint` may appear to be the last statement.

It is important to note that unless threads are used (Section D.4), the call to `update` is not immediate and is often delayed until an event occurs. In this case, code that follows `repaint` appears to be executed prior to the actual execution of the repainting.

D.3.3 Events

Java's original event-handling system was cumbersome and has been completely redone.

When the user uses the mouse or types on the keyboard, the operating system produces an event. Java's original event-handling system was cumbersome and has been completely redone. The new model is much simpler to program than the old. Note that the two models are incompatable: Java 1.1 events are not understood by Java 1.0 compilers and vice versa. The basic rules are as follows:

1. Any class that is willing to provide code to handle an event must `implement` a *listener* interface. Examples of listener interfaces are `ActionListener`, `WindowListener`, and `MouseListener`. As usual, implementing an interface means that all methods of the interface must be defined by the class.
2. An object that is willing to handle the event generated by a component must register its willingness with an *add listener* message sent to the event-generating component. When a component generates an event, the event will be sent to the object that has registered to receive it. If no object has registered to receive it, then it is ignored.

An action event is generated when the user presses a `Button`; it is handled by an `actionListener`.

For an example, consider the action event, which is generated when the user presses a `Button`, hits *Return* while in a `TextField`, or selects from a `List` or `MenuItem`. The simplest way to handle the `Button` click is to have its container implement `ActionListener` by providing an `actionPerformed` method and registering itself with the `Button` as its event handler.

This is shown for our running example in Figure D.1 as follows. Recall that in Figure D.5, we already have done two things. At line 4, `GUI` declares that it implements the `ActionListener`, and at line 10, an instance of `GUI` registers itself as its `Button`'s action event handler. In Figure D.12 (page 748), we implement the listener by having `actionPerformed` call `setParam` in the `GUICanvas` class. This example is simplified by the fact that there is only one