Simple Painter

Description of the Simple Painter program

We build a simple program that will let the user make a simple line drawing (possibly dot-to-dot) and select the color of the lines. This time the end point of the line will be selected by a mouse click in the graphics window. One text field view will track the mouse motion. The New Polygon action will start a new series of lines starting at the next mouse click location.

Simple Painter snapshot:
We only need to see how the mouse adapter is used in the buffered panel, how the actions that respond to mouse actions are defined, and how the mouse location is recorded.

**View**
This program uses two pairs of TextFieldViews to allow the user to select two sets of endpoints for a line. One of the TextFieldViews is set by the mouse tracking action. The ColorView, the TextAreaView, and the BufferedPanel are the same as in the Simple Drawing program.

**Model**
The mouse tracking action uses the coordinates extracted from the mouse location to directly set the view state for its corresponding text field view. The draw action records the coordinates of the first point by extracting the data from the first TextFieldView to a Point2D.Double object. The mouse location that represents the end point of the line is extracted into another Point2D.Double object. These two objects are instantiated every time the draw action is selected. The line is drawn, the history is recorded, the first TextFieldView is set to new values, and the objects die as the action method exits. This represents the Model part of the program.

**Actions**
The program uses one ActionsPanel with two associated Actions. The clear Action clears the history, the graphics window, and resets the TextFieldViews to the original default values used in the constructor. The startNewPolygon Action resets the starting state variable to indicate that the next point selected by a mouse click will be a start point of the next line.

**Mouse Event Actions**
The program has two mouse event actions. The track action continuously extracts the coordinates of the mouse and displays these values by setting its corresponding TextFieldView. The draw action responds to a mouse click. It extracts the mouse location at the time of the click and, as desired, draws a line in currently selected color in the graphics window. It then updates the history and resets the TextFieldView for the starting point of the line to the currently used end point. This represents the Mouse Event Actions part of the program.

**GUI**
The GUI is enclosed in a QuickJPTFrame titled Simple Painter. The SimplePainter extends the DisplayPanel and uses the BorderLayout to organize its appearance. Each pair of TextFieldViews that represent the endpoints of a line is installed in a TablePanel with annotations to the left. Together, these two displays are installed in another TablePanel pointData with horizontal alignment. The TextAreaView that holds the history is installed in a ScrollableDisplay to make sure we can see all of the line coordinates as needed. The ColorView, the actions panel, and the TextAreaView, are combined in a TablePanel controls. The BufferedPanel is installed in a Display titled Graphics and packaged into a TablePanel with pointData to form the right hand
side of the GUI. Finally, the **TablePanel controls**, and the right hand side are installed in the proper locations in the **SimplePainter DisplayPanel**.

The diagram shows how the components fit together - omitting the lowest level text field views:

Notice that there are no panels that activate the mouse actions.

Finally, we add text to the **TextAreaView** as follows:

```java
// show the color and the line coordinates for history
history.append("(" + color.getViewState() + ")\n");
history.append("(" + (int)P1.x + ", " + (int)P1.y + ") : ");
history.append("(" + (int)P2.x + ", " + (int)P2.y + ")\n");
```
Using the BufferedPanel with Mouse Actions

We declare the BufferedPanel as before. In the constructor for the application we specify the actions that respond to mouse events inside of the BufferedPanel as follows:

```java
// get the mouse action adapter from the graphics window panel
MouseActionAdapter adapter = window.getMouseActionAdapter();

// draw if mouse clicked
adapter.addMouseClickedAction(new MouseAction() {
    public void mouseActionPerformed(MouseEvent mevt) {
        draw(mevt);
    }
});

// track mouse motions
adapter.addMouseMovedAction(new MouseAction() {
    public void mouseActionPerformed(MouseEvent mevt) {
        track(mevt);
    }
});
```

The first statement retrieves an adapter object connected to the listener inside of the BufferedPanel. The second statement defines the MouseClickedAction to be our function draw that will get the mouse event as its argument. The third statement defines the track function. The mouse event is passed to these functions as an argument, so that it can be used to extract information about the mouse at the time event was recorded. The track function is very simple:

```java
public void track(MouseEvent mevt) {
    // record the mouse position in its text field view
    xMouseTFV.setViewState((int)mevt.getX() + "");
    yMouseTFV.setViewState((int)mevt.getY() + ");
}
```

Mouse location x is extracted through the mevt.getX() function call. Its return value is then used directly as the first argument for the setState function. The second coordinate is handled similarly.

The draw function has a number of tasks. The following statement is used to extract the mouse location:

```java
// get the end point from mouse location
Point2D.Double P2 =
    new Point2D.Double(mevt.getX(), mevt.getY());
```

All the rest of this function consists of tasks already explained in Simple Drawing program.
JPT Idioms

Mouse Action Adapter

MouseActionAdapter object:

```java
MouseActionAdapter adapter = window.getMouseActionAdapter ();
```

adapter the mouse action adapter identifier
window the panel in which this mouse action adapter is active

Specifying the action when the mouse is clicked:

```java
adapter.addMouseClickedAction(new MouseAction() {
    public void mouseActionPerformed(MouseEvent mevt) {
        draw(mevt);
    }
});
```

adapter the mouse action adapter identifier
draw the method that will be called on a mouse click

Specifying the action when the mouse is moved:

```java
adapter.addMouseMovedAction(new MouseAction() {
    public void mouseActionPerformed(MouseEvent mevt) {
        track(mevt);
    }
});
```

adapter the mouse action adapter identifier
track the method that will be called when the mouse is moved

Extracting the mouse location from a mouse event:

```java
Point2D.Double P2 = new Point2D.Double(mevt.getX(), mevt.getY());
```

mevt the mouse action event passed to the mouse tracking action or mouse clicked action