

## 12 User Interactions

### Goals

In this lab you will learn a little about programming user interactions using the Model-View-Control pattern for organizing the responsibilities.

The JPT library allows you to concentrate on the key concepts and avoid the pitfalls of multitude of details, typically associated with GUI programming.

### The Existing Balloon Controls

Examine briefly the code already supplied with the lab. The code consists of the following classes:

- **class** *Balloon* represents one balloon object, allows the user to move it, paint it, and to compare two balloons for closeness to the top of the graphics window.
- **interface** *BalloonInput* contains two methods: *demandBalloon()* and *requestBalloon()*.
- **class** *ConsoleBalloonInput* implements the *BalloonInput* interface used for reading the input from the console.
- **class** *BalloonInputView* defines a GUI to request the user input for the data needed to initialize one *Balloon* instance. It contains two *TextFieldViews*, one *SliderView*, and one *ColorView*.
- **class** *GUIBalloonInput* implements the *BalloonInput* interface for extracting the user input from the *BalloonInputView* GUI.

Run the code, and note the behavior in response to the various buttons.

### Answer the following questions

1. Read the code for the **class** *Balloon*. Add the method *eraseBalloon* which will paint the balloon in a white color (*Color.white*). Make sure you have the examples and tests for this method.

2. Read the code for the method *testConsoleInput* in the **class** *Interactions*. Describe to your partner (find one, just for this exercise) what the method does. Look at the *ConsoleBalloonInput* class to make sure you understand how the input is being read in.
3. Find the code for the action for the **New** button. Currently, it only sets the value of the *Balloon* instance variables. Add to this action a call to the method which paints the balloon, from the **class** *Balloon*. Make sure it works.
4. Find all places where the *xTFV* is defined or used. Talk with your partner and make sure you understand what is the purpose of each of these code segments.

Using a similar technique, define a new *TextView* named *rTFV*, to represent the numerical value of the *Balloon radius*.

Test the behavior of the slider. Does it have any effect on the balloon? Does it have any effect on the value displayed in the *rTFV* field? Change the value of the *rTFV* field. Does it affect the slider? Does it affect the balloon?

5. Define two new *SimpleActions* and the corresponding methods — an *rTFVaction* and a *SliderAction*. It does not matter what you choose for the label, because we are not going to use the actions with a button.

The first one *void rTFVaction* will be invoked when the value in the field *rTFV* changes. It should then set the value of the balloon *radius* and the value of the *rSlider* to the value displayed in the *rTFV*. To set the state of the *rSlider* use the method

```
rSlider.setViewState(" " + b.radius);
```

The second method *void rSliderAction()* will be invoked every time the location of the slider (and the value it represents) changes. It must then change the *radius* of the balloon and set the view state of the *rTFV* calling the method *setViewState* in a manner similar to the above. If you run the program now, you may be surprised to see that these changes have no effect. Can you think of the way to test that the methods work correctly?

6. Now you have to tell the *rSlider* and the *rTFV* to perform this action when their values change. The following two statements have to be added at the end of the method *void createViews()*:

```
rTFV.addActionListener(rTFVaction);  
rSlider.addSlidingAction(slidingAction);
```

Test that this works.

7. Now that you have seen the method *setViewState*, add such method to the **class** *BalloonInputView*. To see that it works, we need to modify some of the fields of a *Balloon* instance and invoke the method. Try it.
8. In the last part you will control the balloon with the mouse. The first thing you need to do is to change the manner in which the GUI is displayed. Look at the code in the **class** *Interactions* for the method *testBalloonControl()*. Replace the line which calls the method *showOKDialog* with the following:

```
JPTFrame.createQuickJPTFrame(" Balloon Control ", bc);
```

This places the *BalloonControl* into a window that runs *in its own thread*, i.e. independently of the rest of the application. That allows the rest of the application to watch out for the mouse movement and clicks inside of the graphics window.

9. The first mouse action you will build will increase the radius of the balloon by ten, every time you click the mouse. All of this is in the **class** *BalloonControl*. Start by defining the method *protected click(MouseEvent mevt)* which does the following:
- Print into the console a message that the mouse was clicked.
  - Erase the balloon
  - Increase the balloon radius by 10
  - Set the view state of the *BalloonInputView bView* to the current values of the balloon. (Only the radius has changed, but it is easier to let the *BalloonView* do the whole job by invoking the method *setViewState*.)
  - Finally, paint the changed balloon.

10. We need to ask the window to provide us with its *MouseActionAdapter* as follows:

- After the definition of the *BufferedPanel*, add the definition:

```
public MouseActionAdapter mouseAdapter;
```

- Inside of the constructor for the **class** *BalloonControl* first initialize the *mouseAdapter* as follows:

```
mouseAdapter = window.getMouseActionAdapter();
```

- Add the action to perform when the mouse is clicked as follows:

```
// respond to mouse clicks
mouseAdapter.addMouseListener(
    new MouseAction() {
        public void mouseActionPerformed(MouseEvent mevt){
            click(mevt);
        }
    });
```

At this point you should test that your program runs as you expected.

- Finally, you will make the balloon move when the mouse moves. Do all the steps you have done for the clicked action, but do not get a new *mouseAdapter*. The following code will add the action:

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

Inside of the *track* method get the coordinates of the mouse as follows:

```
b.x = mevt.getX();
b.y = mevt.getY();
```

and see what your program does. (Probably nothing - you still have to erase the old balloon, before you make the changes, paint the new balloon, and as a courtesy, set the view state for the view.) Now you should have fun.