# ЗАНЯТИЕ 19
# Получение данных от пользователя

**На этом занятии вы узнаете...**

- как создавать программы, которые реагируют на события;
- как заставить компонент генерировать события;
- как обрабатывать события в программе;
- как хранить информацию в интерфейсе;
- как преобразовывать значения, сохраненные в текстовых полях.

Графический интерфейс, который мы разрабатывали на двух предыдущих занятиях, представляет интерес сам по себе. Пользователи могут щелкать на кнопках, вводить текст в поля и изменять размеры окон. Но рано или поздно даже самый неприхотливый пользователь захочет большего. Графический интерфейс программы должен выполнять определенные действия в ответ на щелчок мыши или ввод данных с клавиатуры.

Все это становится возможным в том случае, когда приложение Java способно реагировать на пользовательские события. Данный процесс называется *обработкой событий* и будет рассмотрен на этом занятии.

## Как научить программу “слушать”

Пользовательское событие в Java происходит тогда, когда пользователь выполняет действие с помощью мыши, клавиатуры или другого устройства ввода.

Прежде чем вы сможете обрабатывать события, следует научить объект слушать. Для реагирования на пользовательские события нужно использовать один или несколько интерфейсов `EventListener`. Интерфейс — это элемент объектно-ориентированного программирования на Java, позволяющий классу наследовать требуемое поведение. Это своего рода контракт, который гарантирует включение в класс определенных методов.

Интерфейс EventListener содержит методы, которые позволяют получать данные определенного типа, вводимые пользователем.

Для подключения интерфейса EventListener требуется соблюдение двух условий. Во-первых, поскольку классы-слушатели входят в пакет java.awt.event, следует обеспечить доступ к этому пакету с помощью следующей инструкции:

```
import java.awt.event.*;
```

Во-вторых, класс должен объявляться с ключевым словом implements, реализуя поддержку одного или нескольких интерфейсов-слушателей. Следующая инструкция создает класс, который реализует интерфейс ActionListener, отвечающий за щелчки на кнопках и пунктах меню:

```
public class Graph implements ActionListener {
```

Интерфейсы EventListener позволяют компонентам графического интерфейса генерировать пользовательские события. В случае отсутствия нужного интерфейса компонент не сможет ничего сделать, чтобы быть услышанным другими частями программы. Программа должна включать интерфейс-слушатель для каждого типа компонента, от которого она хочет получать события. Чтобы программа реагировала на щелчок мыши на кнопке или на нажатие клавиши <Enter> в текстовом поле, следует включить интерфейс ActionListener. Чтобы программа могла реагировать на выбор вариантов из списка или установку флажков, потребуется интерфейс ItemListener.

Если требуется несколько интерфейсов в одном и том же классе, поместите их названия, разделенные запятыми, после ключевого слова implements, как показано в следующем примере.

```
public class Graph3D implements ActionListener, MouseListener {
   // ...
}
```

# Настройка компонентов для прослушивания

После того как реализован интерфейс для определенного компонента, необходимо заставить этот компонент генерировать пользовательские события. Интерфейс ActionListener позволяет реагировать на события действия, такие как щелчок на кнопке или нажатие клавиши <Enter>.

Чтобы объект JButton начал генерировать события, воспользуйтесь методом addActionListener(), как показано ниже.

```
JButton fireTorpedos = new JButton("Пуск торпед");
fireTorpedos.addActionListener(this);
```

Этот код создает кнопку `fireTorpedos` и вызывает метод кнопки `addActionListener()`. Ключевое слово `this`, которое используется в качестве аргумента метода `addActionListener()`, указывает на то, что текущий объект обрабатывает пользовательские события.

**ПРИМЕЧАНИЕ**

Ключевое слово `this` указывает на объект, в котором оно появляется. Если, например, создать класс `LottoMadness` и использовать ключевое слово `this` в одном из его методов, оно будет ссылаться на объект `LottoMadness`, для которого был вызван данный метод.

# Обработка пользовательских событий

Если пользовательское событие генерируется компонентом, у которого есть слушатель, автоматически вызывается метод-обработчик. Этот метод должен содержаться в классе, указанном при подключении слушателя к компоненту.

Каждый слушатель включает разные методы, которые вызываются для реагирования на соответствующие события. Интерфейс `ActionListener` отсылает события методу `actionPerformed()`, который определяется так.

```
public void actionPerformed(ActionEvent event) {
   // Тело метода
}
```

Все события, полученные программой, передаются этому методу. Если только один компонент генерирует события действий, в метод включаются инструкции обработки полученного события. Если же события могут генерироваться несколькими компонентами, нужно проверить объект, переданный методу.

Аргументом метода `actionPerformed()` является объект `ActionEvent`. Пользовательские события, которые могут генерироваться в программе, представляются несколькими классами. Эти классы содержат методы, позволяющие определить компонент, который вызвал событие. Если объект `ActionEvent` в методе `actionPerformed()` называется `event`, то можно идентифицировать компонент с помощью следующей инструкции:

```
String cmd = event.getActionCommand();
```

Метод `getActionCommand()` возвращает строку. Если компонент является кнопкой, строка будет надписью на этой кнопке. Если же компонент

является текстовым полем, строка будет текстом, введенным в поле. Метод `getSource()` возвращает сам объект, сгенерировавший событие.

Можно использовать следующий метод `actionPerformed()` для получения событий от трех компонентов: кнопки `start` (объект `JButton`), текстового поля `speed` (объект `JTextField`) и еще одного текстового поля `viscosity`.

```
public void actionPerformed(ActionEvent event) {
   Object source = event.getSource();
   if (source == speed) {
      // Событие сгенерировано полем speed
   } else if (source == viscosity) {
      // Событие сгенерировано полем viscosity
   } else {
      // Событие сгенерировано кнопкой start
   }
}
```

Вызывайте метод `getSource()` для всех пользовательских событий, чтобы идентифицировать конкретный объект, который сгенерировал событие.

## События флажка и поля со списком

Поля со списком и флажки требуют наличия интерфейса `ItemListener`. Вызовите метод компонента `addItemListener()`, чтобы заставить его генерировать события. Следующие инструкции создают флажок `superSize`, который генерирует пользовательские события в случае его выбора или отмены выбора.

```
JCheckBox superSize = new JCheckBox("Super Size", true);
superSize.addItemListener(this);
```

Эти события перехватываются методом `itemStateChanged()`, которому в качестве аргумента передается объект `ItemEvent`. Если требуется узнать, какой объект сгенерировал событие, следует вызвать метод `getItem()` объекта события.

Чтобы определить, был ли флажок выбран или сброшен, сравните значение, возвращаемое методом `getStateChange()`, с константой `ItemEvent.SELECTED` или `ItemEvent.DESELECTED`.

```
public void itemStateChanged(ItemEvent item) {
   int status = item.getStateChange();
   if (status == ItemEvent.SELECTED) {
      // Элемент выбран
   }
}
```

Чтобы определить, какое значение было выбрано в объекте `JComboBox`, используйте метод `getItem()` и преобразуйте полученное значение в строку.

```
Object which = item.getItem();
String answer = which.toString();
```

## События клавиатуры

Если программа должна реагировать сразу же после нажатия клавиши, необходимо обрабатывать события клавиатуры и использовать интерфейс `KeyListener`.

В первую очередь необходимо зарегистрировать компонент, получающий код нажатой клавиши, вызвав его метод `addKeyListener()`. Аргументом метода должен быть объект, реализующий интерфейс `KeyListener`. Если это текущий класс, используйте аргумент `this`.

Объект, который обрабатывает события клавиатуры, должен реализовывать следующие три метода.

- `keyPressed(`*`СобытиеКлавиатуры`*`)`. Вызывается в момент нажатия клавиши.
- `keyReleased(`*`СобытиеКлавиатуры`*`)`. Вызывается в момент отпускания клавиши.
- `keyTyped(`*`СобытиеКлавиатуры`*`)`. Вызывается после того, как клавиша была нажата и отпущена.

Эти методы не возвращают никаких значений (`void`), и аргументом каждого из них является объект `KeyEvent`, который содержит методы, позволяющие получить дополнительную информацию о событии. Например, с помощью метода `getKeyChar()` можно узнать о том, какая клавиша была нажата. Он возвращает значение типа `char`, которое может содержать только буквы, цифры и знаки пунктуации.

Если нужно отслеживать нажатия произвольных клавиш, включая клавиши <Enter>, <Home>, <Page Up> и <Page Down>, вызовите метод `getKeyCode()`, который возвращает целочисленный код клавиши. Далее можно вызвать метод `getKeyText()`, указав полученный код в качестве аргумента. Это позволит получить объект `String`, содержащий название клавиши (например, `Home`, `F1` и т.п.).

В листинге 19.1 содержится код приложения Java, которое идентифицирует нажатые клавиши с помощью метода `getKeyChar()`. Приложение реализует интерфейс `KeyListener`, поэтому содержит методы `keyTyped()`, `keyPressed()` и `keyReleased()`. Но единственный метод, который выполняет реальные действия, — это `keyTyped()` (строки 26–29). Создайте новый класс `KeyViewer`, включите его в пакет `com.java24hours`, введите код из листинга 19.1 и сохраните полученный файл `KeyViewer.java`.

**ЛИСТИНГ 19.1.** Исходный код программы `KeyViewer.java`

```
 1: package com.java24hours;
 2:
 3: import javax.swing.*;
 4: import java.awt.event.*;
 5: import java.awt.*;
 6:
 7: public class KeyViewer extends JFrame implements KeyListener {
 8:    JTextField keyText = new JTextField(80);
 9:    JLabel keyLabel = new JLabel("Нажмите любую клавишу
10:                                  в текстовом поле.");
11:
12:    public KeyViewer() {
13:       super("KeyViewer");
14:       setLookAndFeel();
15:       setSize(350, 100);
16:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17:       keyText.addKeyListener(this);
18:       BorderLayout bord = new BorderLayout();
19:       setLayout(bord);
20:       add(keyLabel, BorderLayout.NORTH);
21:       add(keyText, BorderLayout.CENTER);
22:       setVisible(true);
23:    }
24:
25:    @Override
26:    public void keyTyped(KeyEvent input) {
27:       char key = input.getKeyChar();
28:       keyLabel.setText("Вы нажали " + key);
29:    }
30:
31:    @Override
32:    public void keyPressed(KeyEvent txt) {
33:       // Ничего не делать
34:    }
35:
36:    @Override
37:    public void keyReleased(KeyEvent txt) {
38:       // Ничего не делать
39:    }
40:
41:    private void setLookAndFeel() {
42:       try {
43:          UIManager.setLookAndFeel(
44:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
45:          );
46:       } catch (Exception exc) {
47:          // Игнорировать ошибку
```

```
48:        }
49:     }
50:
51:     public static void main(String[] arguments) {
52:        KeyViewer frame = new KeyViewer();
53:     }
54: }
```

После запуска приложения появится окно, показанное на рис. 19.1. Приложение реализует три метода интерфейса `KeyListener` (строки 25–39), два из которых являются пустыми. Они не нужны самому приложению, но должны формально включаться в рамках контракта по реализации интерфейса, заключенного в строке 7 с помощью ключевого слова `implements`.


**РИС. 19.1.** Обработка событий клавиатуры в программе

## Активизация и отключение компонентов

Наверняка вам не раз доводилось видеть неактивные программные компоненты, окрашенные в серый цвет. Подобное окрашивание означает, что пользователи ничего не могут сделать с компонентом, поскольку он не активен. Активизация и отключение компонентов осуществляются с помощью метода компонента `setEnabled()`. В качестве аргумента метода используется булево значение, поэтому вызов `setEnabled(true)` активизирует компонент, а `setEnabled(false)` отключает его.

Следующие инструкции создают кнопки с надписями Назад, Далее и Готово и отключают первую кнопку.

```
JButton previousButton = new JButton("Назад");
JButton nextButton = new JButton("Далее");
JButton finishButton = new JButton("Готово");
previousButton.setEnabled(false);
```

Это эффективный способ предотвратить генерирование пользовательских событий компонентом. Например, если создается приложение, которое собирает адреса пользователей с помощью текстовых полей, можно отключить кнопку Сохранить адрес до тех пор, пока пользователь не введет адрес, название города и области, а также почтовый индекс.

# Завершенное графическое приложение

Чтобы проиллюстрировать применение классов обработки событий Swing, мы вернемся к приложению `LottoMadness` (см. занятие 18), имитирующему лотерею.

В данный момент приложение `LottoMadness` — это просто графический интерфейс, и если вы будете щелкать на кнопках и вводить текст в поля, в ответ ничего не произойдет. На этом занятии мы создадим новый класс `LottoEvent`, который получает пользовательские данные, проводит розыгрыш лотереи и отслеживает количество выигрышей. После создания класса `LottoEvent` мы добавим несколько строк кода в приложение `LottoMadness`, чтобы задействовать этот класс. Зачастую бывает удобно разделить проекты Swing таким образом, чтобы графический интерфейс находился в одном классе, а методы обработки событий — в другом.

Назначение нашего приложения — оценить шанс пользователя угадать шесть чисел в лотерее на протяжении всей его жизни. На рис. 19.2 показан снимок экрана выполняющейся программы.

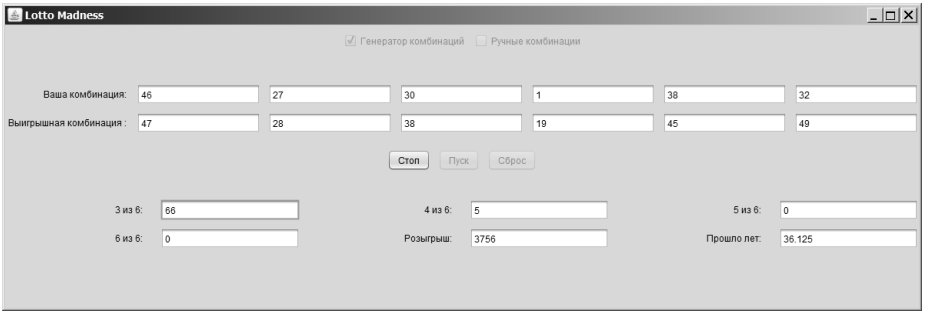


**РИС. 19.2.** Выполняющееся приложение `LottoMadness`

Вместо того чтобы решить эту задачу с помощью инструментов теории вероятности, программа быстро перебирает лотерейные комбинации до тех пор, пока не найдет выигрышную. Поскольку выигрышная комбинация "6 из 6" встречается чрезвычайно редко, программа отображает сообщения о любой выигрышной комбинации, состоящей из трех, четырех или пяти чисел.

Интерфейс включает 12 текстовых полей, предназначенных для чисел лотереи, и два флажка, которые называются `Генератор комбинаций` и `Ручные комбинации`. Шесть текстовых полей, заблокированных для ввода чисел, служат для отображения выигрышных номеров каждой комбинации. Оставшиеся шесть текстовых полей предназначены для ввода номеров лотереи пользователем. При установке флажка Генератор комбинаций выбирается шесть случайных чисел. Если же установлен флажок Ручные комбинации, то пользователь может ввести лотерейные номера вручную.

Пользователь управляет действиями программы с помощью трех кнопок: Стоп, Пуск и Сброс. После щелчка на кнопке Пуск программа запускает поток `playing` и генерирует лотерейные комбинации. После щелчка на кнопке Стоп выполнение потока прекращается, а после щелчка на кнопке Сброс очищаются все поля, чтобы пользователь мог все ввести заново (подробнее о потоках см. занятие 15).

Класс `LottoEvent` реализует три интерфейса: `ActionListener`, `ItemListener` и `Runnable`. Интерфейс `Runnable` связан с потоками и был рассмотрен на занятии 15. Слушатели требуются для обработки пользовательских событий, генерируемых кнопками и флажками приложения. Программе не нужно обрабатывать события, связанные с текстовыми полями, поскольку они применяются исключительно для хранения номеров, выбранных пользователем. Подобное хранение реализуется автоматически.

К программе необходимо подключить основной пакет Swing, `javax.swing`, и пакет обработки событий Java, `java.awt.event`.

Класс содержит две переменные экземпляра:

- `gui`, объект `LottoMadness`;
- `playing`, объект `Thread`, используемый для проведения непрерывного розыгрыша лотереи.

Переменная `gui` применяется для взаимодействия с объектом `LottoMadness`, который содержит графический интерфейс программы. Если нужно внести какие-то изменения в интерфейс или извлечь значение из одного из текстовых полей, используются переменные экземпляра объекта `gui`.

Например, переменная `play` объекта `LottoMadness` представляет кнопку Пуск. Чтобы отключить эту кнопку в объекте `LottoEvent`, воспользуйтесь такой инструкцией:

```
gui.play.setEnabled(false);
```

Чтобы извлечь содержимое поля `got3` (объект `JTextField`), введите следующую инструкцию:

```
String got3value = gui.got3.getText();
```

В листинге 19.2 содержится полный код класса `LottoEvent`. Создайте новый класс, включите его в пакет `com.java24hours`, введите код листинга и сохраните полученный файл `LottoEvent.java`.

**ЛИСТИНГ 19.2.** Исходный код программы `LottoEvent.java`

```
 1: package com.java24hours;
 2:
 3: import javax.swing.*;
```

```
 4: import java.awt.event.*;
 5:
 6: public class LottoEvent implements ItemListener,
 7:                                    ActionListener, Runnable {
 8:
 9:    LottoMadness gui;
10:    Thread playing;
11:
12:    public LottoEvent(LottoMadness in) {
13:       gui = in;
14:    }
15:
16:    @Override
17:    public void actionPerformed(ActionEvent event) {
18:       String command = event.getActionCommand();
19:       if (command.equals("Пуск")) {
20:          startPlaying();
21:       }
22:       if (command.equals("Стоп")) {
23:          stopPlaying();
24:       }
25:       if (command.equals("Сброс")) {
26:          clearAllFields();
27:       }
28:    }
29:
30:    void startPlaying() {
31:       playing = new Thread(this);
32:       playing.start();
33:       gui.play.setEnabled(false);
34:       gui.stop.setEnabled(true);
35:       gui.reset.setEnabled(false);
36:       gui.quickpick.setEnabled(false);
37:       gui.personal.setEnabled(false);
38:    }
39:
40:    void stopPlaying() {
41:       gui.stop.setEnabled(false);
42:       gui.play.setEnabled(true);
43:       gui.reset.setEnabled(true);
44:       gui.quickpick.setEnabled(true);
45:       gui.personal.setEnabled(true);
46:       playing = null;
47:    }
48:
49:    void clearAllFields() {
50:       for (int i = 0; i < 6; i++) {
51:          gui.numbers[i].setText(null);
52:          gui.winners[i].setText(null);
```

```
53:       }
54:       gui.got3.setText("0");
55:       gui.got4.setText("0");
56:       gui.got5.setText("0");
57:       gui.got6.setText("0");
58:       gui.drawings.setText("0");
59:       gui.years.setText("0");
60:    }
61:
62:    @Override
63:    public void itemStateChanged(ItemEvent event) {
64:       Object item = event.getItem();
65:       if (item == gui.quickpick) {
66:          for (int i = 0; i < 6; i++) {
67:             int pick;
68:             do {
69:                pick = (int) Math.floor(Math.random() * 50 + 1);
70:             } while (numberGone(pick, gui.numbers, i));
71:             gui.numbers[i].setText("" + pick);
72:          }
73:       } else {
74:          for (int i = 0; i < 6; i++) {
75:             gui.numbers[i].setText(null);
76:          }
77:       }
78:    }
79:
80:    void addOneToField(JTextField field) {
81:       int num = Integer.parseInt("0" + field.getText());
82:       num++;
83:       field.setText("" + num);
84:    }
85:
86:    boolean numberGone(int num, JTextField[] pastNums, int count) {
87:       for (int i = 0; i < count; i++) {
88:          if (Integer.parseInt(pastNums[i].getText()) == num) {
89:             return true;
90:          }
91:       }
92:       return false;
93:    }
94:
95:    boolean matchedOne(JTextField win, JTextField[] allPicks) {
96:       for (int i = 0; i < 6; i++) {
97:          String winText = win.getText();
98:          if ( winText.equals( allPicks[i].getText() ) ) {
99:             return true;
100:         }
101:      }
```

```
102:       return false;
103:    }
104:
105:    @Override
106:    public void run() {
107:       Thread thisThread = Thread.currentThread();
108:       while (playing == thisThread) {
109:          addOneToField(gui.drawings);
110:          int draw = Integer.parseInt(gui.drawings.getText());
111:          float numYears = (float) draw / 104;
112:          gui.years.setText("" + numYears);
113:
114:          int matches = 0;
115:          for (int i = 0; i < 6; i++) {
116:             int ball;
117:             do {
118:                ball = (int) Math.floor(Math.random() * 50 + 1);
119:             } while (numberGone(ball, gui.winners, i));
120:             gui.winners[i].setText("" + ball);
121:             if (matchedOne(gui.winners[i], gui.numbers)) {
122:                matches++;
123:             }
124:          }
125:          switch (matches) {
126:             case 3:
127:                addOneToField(gui.got3);
128:                break;
129:             case 4:
130:                addOneToField(gui.got4);
131:                break;
132:             case 5:
133:                addOneToField(gui.got5);
134:                break;
135:             case 6:
136:                addOneToField(gui.got6);
137:                gui.stop.setEnabled(false);
138:                gui.play.setEnabled(true);
139:                playing = null;
140:          }
141:          try {
142:             Thread.sleep(100);
143:          } catch (InterruptedException e) {
144:             // Ничего не делать
145:          }
146:       }
147:    }
148: }
```

Класс `LottoEvent` содержит единственный конструктор `LottoEvent (LottoMadness)`. Объект `LottoMadness`, указанный в качестве аргумента, идентифицирует объект, который полагается на объект `LottoEvent` для обработки пользовательских событий и проведения розыгрышей лотереи.

В классе используются следующие методы.

- `clearAllFields()`. Очищает все текстовые поля приложения. Вызывается после щелчка на кнопке Сброс.
- `addOneToField()`. Преобразует содержимое текстового поля в целочисленное значение, увеличивает это значение на единицу, а затем преобразует его обратно в содержимое текстового поля. Поскольку все текстовые поля хранятся в виде строк, нужно выполнять дополнительные действия, чтобы использовать их в выражениях.
- `numberGone()`. Имеет три аргумента: число из розыгрыша лотереи, массив объектов `JTextField` и целочисленная переменная `count`. Метод исключает дублирование номеров при проведении розыгрыша лотереи.
- `matchedOne()`. Имеет два аргумента: объект `JTextField` и массив из шести объектов `JTextField`. Метод проверяет, соответствует ли один из номеров, введенных пользователем, номерам из текущего розыгрыша лотереи.

Метод `actionPerformed()` получает объект события, сгенерированного после щелчка пользователя на кнопке. Применяемый в нем метод `getActionCommand()` возвращает надпись на кнопке. Это позволяет определить, на какой из кнопок был выполнен щелчок.

При щелчке на кнопке Пуск вызывается метод `startPlaying()`, отключающий четыре компонента. При щелчке на кнопке Стоп вызывается метод `stopPlaying()`, который активизирует каждый компонент, за исключением кнопки Стоп.

Метод `itemStateChanged()` обрабатывает пользовательские события, вызванные установкой флажков Генератор комбинаций и Ручные комбинации. Вызываемый в нем метод `getItem()` возвращает объект, представляющий флажок, на котором был выполнен щелчок мышью. Если был установлен флажок Генератор комбинаций, лотерейным номерам пользователя присваиваются шесть случайных значений, находящихся в диапазоне от 1 до 50. В противном случае текстовые поля, в которых хранятся лотерейные номера, будут очищены.

Класс `LottoEvent` использует номера от 1 до 50 для каждого шара в лотерейном розыгрыше. Эта методика реализуется в строке 118, в которой значение, возвращаемое методом `Math.random()`, умножается на 50, к результату добавляется 1, и полученное значение используется в качестве аргумента

метода `Math.floor()`. Конечный результат представляет собой случайное число в диапазоне от 1 до 50. Если заменить число 50 другим значением здесь и в строке 69, то можно будет использовать приложение `LottoMadness` для лотерейных розыгрышей с бо́льшим или меньшим диапазоном значений.

В приложении `LottoMadness` отсутствуют переменные, используемые для отслеживания количества лотерейных розыгрышей, выигрышей и текстовых полей с лотерейными номерами. Вместо этого интерфейс сам хранит значения и автоматически отображает их.

Повторно откройте в NetBeans файл `LottoMadness.java`. Чтобы это приложение могло использовать класс `LottoEvent`, необходимо добавить в него всего лишь шесть строк кода.

Прежде всего добавьте новую переменную экземпляра, которая будет хранить объект `LottoEvent`.

```
LottoEvent lotto = new LottoEvent(this);
```

Далее в конструкторе `LottoMadness()` вызовите методы `addItemListener()` и `addActionListener()` для каждого компонента интерфейса, который принимает данные, вводимые пользователем.

```
// Добавить слушателей событий
quickpick.addItemListener(lotto);
personal.addItemListener(lotto);
stop.addActionListener(lotto);
play.addActionListener(lotto);
reset.addActionListener(lotto);
```

Эти инструкции следует добавить в конструктор сразу же после вызова метода `setLayout()`, который задает менеджер компоновки `GridLayout` для фрейма приложения.

В листинге 19.3 содержится полный код приложения `LottoMadness.java` после внесения изменений. Добавленные строки выделены серым цветом, остальные строки не изменились.

**ЛИСТИНГ 19.3.** Исходный код программы `LottoMadness.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5:
 6: public class LottoMadness extends JFrame {
 7:     LottoEvent lotto = new LottoEvent(this);
 8:
 9:     // Настройка строки 1
10:     JPanel row1 = new JPanel();
```

```
11:    ButtonGroup option = new ButtonGroup();
12:    JCheckBox quickpick = new JCheckBox("Генератор комбинаций ",
13:                                        false);
14:    JCheckBox personal = new JCheckBox("Ручные комбинации", true);
15:    // Настройка строки 2
16:    JPanel row2 = new JPanel();
17:    JLabel numbersLabel = new JLabel("Ваша комбинация: ",
18:                                     JLabel.RIGHT);
19:    JTextField[] numbers = new JTextField[6];
20:    JLabel winnersLabel = new JLabel("Выигрышная комбинация: ",
21:                                     JLabel.RIGHT);
22:    JTextField[] winners = new JTextField[6];
23:    // Настройка строки 3
24:    JPanel row3 = new JPanel();
25:    JButton stop = new JButton("Стоп");
26:    JButton play = new JButton("Пуск");
27:    JButton reset = new JButton("Сброс");
28:    // Настройка строки 4
29:    JPanel row4 = new JPanel();
30:    JLabel got3Label = new JLabel("3 из 6: ", JLabel.RIGHT);
31:    JTextField got3 = new JTextField("0");
31:    JLabel got4Label = new JLabel("4 из 6: ", JLabel.RIGHT);
33:    JTextField got4 = new JTextField("0");
34:    JLabel got5Label = new JLabel("5 из 6: ", JLabel.RIGHT);
35:    JTextField got5 = new JTextField("0");
36:    JLabel got6Label = new JLabel("6 из 6: ", JLabel.RIGHT);
37:    JTextField got6 = new JTextField("0", 10);
38:    JLabel drawingsLabel = new JLabel("Розыгрыш: ", JLabel.RIGHT);
39:    JTextField drawings = new JTextField("0");
40:    JLabel yearsLabel = new JLabel("Прошло лет: ", JLabel.RIGHT);
41:    JTextField years = new JTextField("0");
42:
43:    public LottoMadness() {
44:       super("Lotto Madness");
45:
46:       setSize(550, 400);
47:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
48:       GridLayout layout = new GridLayout(5, 1, 10, 10);
49:       setLayout(layout);
50:
51:        // Добавить слушателей
52:        quickpick.addItemListener(lotto);
53:        personal.addItemListener(lotto);
54:        stop.addActionListener(lotto);
55:        play.addActionListener(lotto);
56:        reset.addActionListener(lotto);
57:
58:       FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
59:                                           10, 10);
```

```
60:       option.add(quickpick);
61:       option.add(personal);
62:       row1.setLayout(layout1);
63:       row1.add(quickpick);
64:       row1.add(personal);
65:       add(row1);
66:
67:       GridLayout layout2 = new GridLayout(2, 7, 10, 10);
68:       row2.setLayout(layout2);
69:       row2.add(numbersLabel);
70:       for (int i = 0; i < 6; i++) {
71:          numbers[i] = new JTextField();
72:          row2.add(numbers[i]);
73:       }
74:       row2.add(winnersLabel);
75:       for (int i = 0; i < 6; i++) {
76:          winners[i] = new JTextField();
77:          winners[i].setEditable(false);
78:          row2.add(winners[i]);
79:       }
80:       add(row2);
81:
82:       FlowLayout layout3 = new FlowLayout(FlowLayout.CENTER,
83:                                           10, 10);
84:       row3.setLayout(layout3);
85:       stop.setEnabled(false);
86:       row3.add(stop);
87:       row3.add(play);
88:       row3.add(reset);
89:       add(row3);
90:
91:       GridLayout layout4 = new GridLayout(2, 3, 20, 10);
92:       row4.setLayout(layout4);
93:       row4.add(got3Label);
94:       got3.setEditable(false);
95:       row4.add(got3);
96:       row4.add(got4Label);
97:       got4.setEditable(false);
98:       row4.add(got4);
99:       row4.add(got5Label);
100:      got5.setEditable(false);
101:      row4.add(got5);
102:      row4.add(got6Label);
103:      got6.setEditable(false);
104:      row4.add(got6);
105:      row4.add(drawingsLabel);
106:      drawings.setEditable(false);
107:      row4.add(drawings);
108:      row4.add(yearsLabel);
```

```
109:        years.setEditable(false);
110:        row4.add(years);
111:        add(row4);
112:
113:        setVisible(true);
114:     }
115:
116:     private static void setLookAndFeel() {
117:        try {
118:           UIManager.setLookAndFeel(
119:              "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
120:           );
121:        } catch (Exception exc) {
122:           // Игнорировать ошибки
123:        }
124:     }
125:
126:     public static void main(String[] arguments) {
127:        LottoMadness.setLookAndFeel();
128:        LottoMadness frame = new LottoMadness();
129:     }
130: }
```

После добавления выделенных строк запустите приложение, чтобы оценить свои шансы на выигрыш в лотерею. Как и следовало ожидать, игра в лотерею — бесполезное занятие. Шанс угадать все шесть номеров настолько мал, что вам и жизни не хватит, даже если вы будете жить так долго, как библейские патриархи.

**ПРИМЕЧАНИЕ**

После запуска программы `LottoMadness` было просчитано 410 732 244 возможных розыгрыша, что заняло бы около 3,9 млн лет при условии проведения розыгрышей дважды в неделю. В результате было определено 6 364 880 победителей, угадавших 3 из 6 номеров, 337 285 победителей, которые угадали 4 из 6 номеров, 6 476 победителей, которые угадали 5 из 6 номеров, и 51 победитель, который угадал 6 из 6 номеров (приблизительно один победитель на каждые 8 млн розыгрышей). Первый выигрыш был получен после участия в 241 225 розыгрышах, на что было потрачено почти 2 320 лет.

# Резюме

Благодаря библиотеке Swing можно создавать профессионально выглядящие программы, прилагая для этого совсем небольшие усилия. Несмотря на то что приложение `LottoMadness` больше по размерам, чем многие из примеров, которые были рассмотрены на предыдущих занятиях, в действительности половина кода этой программы состоит из инструкций, предназначенных для создания интерфейса.

Если вы какое-то время поработаете с приложением, имитирующим проведение лотереи, то станете еще больше завидовать удаче людей, которые таки угадали заветные шесть номеров. Когда я последний раз запустил программу, я пришел к выводу, что мне придется потратить 17 000 долл. и лучшие 165 лет моей жизни, чтобы один раз угадать 5 номеров из 6, 10 раз угадать 4 номера из 6 и 264 раза угадать 3 номера из 6. По сравнению с этими сомнительными выгодами шансы улучшить навыки программирования на Java после прочтения этой книги близки к 100%.

# Вопросы и ответы

**В. Нужно ли вызывать метод `paint()` или `repaint()`, чтобы указать на изменение текстового поля?**

**О.** После использования метода `setText()` для изменения значения текстового компонента вам не придется делать еще что-то. В подобном случае все необходимые действия автоматически выполняются в Swing.

**В. Почему так часто приходится импортировать класс и один из его подклассов, как в листинге 19.1, где выполнялись инструкции `import java.awt.*` и `java.awt.event.*`? Может ли первая из этих инструкций замещать вторую?**

**О.** Несмотря на то что имена пакетов `java.awt` и `java.awt.event` подразумевают их родство, в Java сами пакеты не наследуются. Один пакет не может входить в состав другого.

В случае использования символа звездочки в инструкции импорта все классы пакета становятся доступными в программе. Но это касается только имен классов, а не пакетов. Одиночная инструкция `import` загружает классы из одного пакета.

# Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

## Контрольные вопросы

1. Почему события действия называются именно так?
   - **А.** Они происходят в ответ на что-либо.
   - **Б.** Они указывают на то, что в ответ следует предпринять какое-то действие.
   - **В.** Они весьма действенны.

2. Что означает ключевое слово `this` в качестве аргумента метода `addActionListener()`?

   **А.** Для обработки события должен применяться текущий слушатель.

   **Б.** Текущее событие получает приоритет над другими событиями.

   **В.** События будут обрабатываться текущим объектом.

3. Какой текстовый компонент хранит вводимые пользователем данные в виде целых чисел?

   **А.** `JTextField`.

   **Б.** `JTextArea`.

   **В.** Ни А, ни Б.

## Ответы

1. **Б.** События действий включают щелчки на кнопке и выбор элемента в раскрывающемся меню.
2. **В.** Ключевое слово `this` ссылается на текущий объект. Если вместо `this` указать имя объекта, то этот объект будет перехватывать события и, как ожидается, обрабатывать их.
3. **В.** Компоненты `JTextField` и `JTextArea` хранят свои значения в виде текста, поэтому, прежде чем использовать их в качестве целочисленных значений, чисел с плавающей точкой или других нетекстовых значений, необходимо выполнить соответствующее преобразование.

## Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Добавьте в приложение `LottoMadness` текстовое поле, которое используется совместно с инструкцией `Thread.sleep()` в классе `LottoEvent`, чтобы замедлить скорость проведения розыгрышей.
- Измените проект `LottoMadness` таким образом, чтобы можно было угадать пять номеров из диапазона 1–90.